JSC 38605

Guidelines for the Rapid Development of Software Systems

ENGINEERING DIRECTORATE

AEROSCIENCE AND FLIGHT MECHANICS DIVISION

2 December 1996

National Aeronautics and
Space Administration

Lyndon B. Johnson Space Center
Houston, TX

JSC 38605

Guidelines for the Rapid Development of Software Systems

Prepared By:

Denise M. DiFilippo
G. B. Tech, Incorporated

Bill G. Brown
SysComm Development

Approved By:

David A. Petri
GN&C Rapid Development Lab Manager
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

James P. Ledet
Code Q RTOP Project Manager
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Concurred By:

Aldo J. Bordano, Chief
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Sonya F. Sepahban, Deputy Chief
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Preface

This document captures the results of work performed in FY'96 with funds provided under the Research and Technology Operation Plan (RTOP) by the Office of Safety and Mission Assurance (OSMA). OSMA has delegated requirements for the Agency Software Program to Ames Research Center Software Technology Division (ARC/IT) located in Fairmont, West Virginia. Work under this initiative was managed at ARC/IT by Kathryn M. Kemp, Deputy Chief, Software Technology Division, and George J. Sabolish, Center Software Initiative Manager. The work was performed in the Aeroscience and Flight Mechanics Division at the Johnson Space Center in collaboration with the Jet Propulsion Laboratory.

The results of FY'96 work are documented in a 2 volume set consisting of:

- JSC 38605 Guidelines for the Rapid Development of Software Systems
- JSC 38606 Guidelines for the Rapid Development of Software Systems - References

This initiative continues in FY'97 with the objective of determining the effectiveness of the guidelines by using them in a rapid software development demonstration project. The results of the demonstration project will be documented along with any refinement to these guidelines.

Contributors

The following people have contributed to the ideas and content of this document: Bill Brown(Syscom), Faye Budlong (Draper Labs), John Craft(EG2), Denise DiFilippo(GBTech), Jim Ledet(EG4), Harry Ohls(JPL), Bill Othon(Lincom), Doug Pesek(McDonnell Douglas), Dave Petri(EG1), Sonya Sepabahn(EG1), Glenn Venables(McDonnell Douglas), Brad Wissinger(McDonnell Douglas), Doug Zimpfer(Draper Labs).

Table of Contents

Figures

Tables

Acronyms and Abbreviations

ACRV	Assured Crew Return Vehicle
AFMD	Aeroscience and Flight Mechanics Division
COTS	commercial, off-the-shelf
CMM	Capability Maturity Model
DOF	degree-of-freedom
GN&C	Guidance, Navigation & Control
HIL	hardware-in-the-loop
IV&V	Independent Verification and Validation
ISI	Integrated Systems Inc.
ISSA	International Space Station Alpha
JPL	Jet Propulsion Laboratory
JSC	Johnson Spaceflight Center
MDA-HD	McDonnell Douglas Aerospace – Houston Division
MOD	Mission Operations Directorate
OIL	Operator-in-the-loop
RDL	Rapid Development Laboratory
RTOP	Research and Technology Objectives and Plan
SEI	Software Engineering Institute

1.0 Overview

The Aeroscience and Flight Mechanics Division (AFMD) at the National Aeronautics and Space Administration-Johnson Space Center (NASA-JSC) Engineering Directorate is exploring ways of producing Guidance, Navigation & Control (GN&C) systems more efficiently and effectively. A significant portion of this effort is software development, integration, testing and verification.

To achieve these goals, the AFMD established the GN&C Rapid Development Laboratory (RDL), a hardware/software facility designed to take a GN&C design project from initial inception through hardware-in-the-loop (HIL) testing and perform final GN&C system verification. The operations approach for the RDL concentrated on the use of commercial, off-the-shelf (COTS) software products to develop the GN&C algorithms in the form of graphical data flow diagrams, to automatically generate source code from these diagrams and to run in a real-time, HIL environment under a Rapid Development paradigm.

The success of these efforts has motivated further study and documentation of Rapid Development methodologies. The initial goal was to formalize the successful methods used to date in the GN&C RDL. Subsequently the team expanded on these methods, based on knowledge gained from extensive search and study of the current literature. The resulting methodology is documented here as a guidebook for Rapid Development. The methodology will be tested, via storyboarding, prototyping, application to several major new design projects (e.g. X-38, Orbiter Upgrades), peer reviews, and test cases. The process will be refined as lessons are learned. This is in step with the overall philosophy of Rapid Development: to revise the plan, based on lessons learned, before moving on, thus reducing risk by finding problems early in the development cycle. The team welcomes comments and feedback, especially any observations from those who have practical experience using this or a similar methodology.

2.0 Introduction

What is the best way to develop systems that include software as a significant component? For many years, the “gold standard” of software development has been the use of structured analysis and programming in the context of the “waterfall” model of a system life-cycle (see Figure 1. The Traditional Waterfall Flight Software Development Approach on page 3). In recent years, a modification to the waterfall model (Incremental Development) which partitions large systems into independent deliverables and then sequentially applies the waterfall model to each subset, has gained popularity.

2.1 Motivation for a new methodology

In the context of many of today’s systems problems, the waterfall model approach to system development, and its modified incremental development approach, are often ineffective for a variety of reasons.

As system complexity increases, it becomes more difficult to completely specify detailed requirements in text form. The documents that attempt to describe these systems become large and complex. The requirements may interact in intricate and complex ways. The review and sign-off processes can be lengthy and expensive. Verifying that the requirements documentation is complete, accurate and consistent can be a daunting or impossible task.

As the problems to be addressed increase in complexity, the solution approaches become less obvious. It may not be reasonable to ask a user community to enumerate requirements, since technology may be able to offer approaches never before used. That is, we have gone beyond using software to just duplicate human effort faster. Still, software developers usually will not be experts in the domain of the problems to be solved, so it is similarly unrealistic to depend solely on them to define a system. A cooperative effort, among domain experts and technology experts, to discover system requirements can leverage the value added of new systems. The waterfall methodology often does not accommodate this philosophy, since requirements are developed independently and “thrown over the wall” to developers who may have no knowledge of the system beyond that written in the requirements documents.

The pace of change coupled with the potentially long lead time to develop systems often creates the dilemma of today’s new systems meeting yesterday’s requirements. This is especially true when system requirements are completely and contractually specified and fixed early in the development cycle. If the requirements are handed off to developers who are completely separate from the domain experts, then developers are likely to be unaware of important changes that occur during the development cycle. This effect is compounded when the problem domain exists in an area where the state-of-the-art is changing at a rapid pace.

Ultimately, the need for a better way to develop software systems is driven by the need to manage the risks involved. These include development costs, maintenance costs, and the more difficult to measure cost to an organization when it does not have the best system for the customers’ needs. The bottom line is that we need better, faster, cheaper software systems.

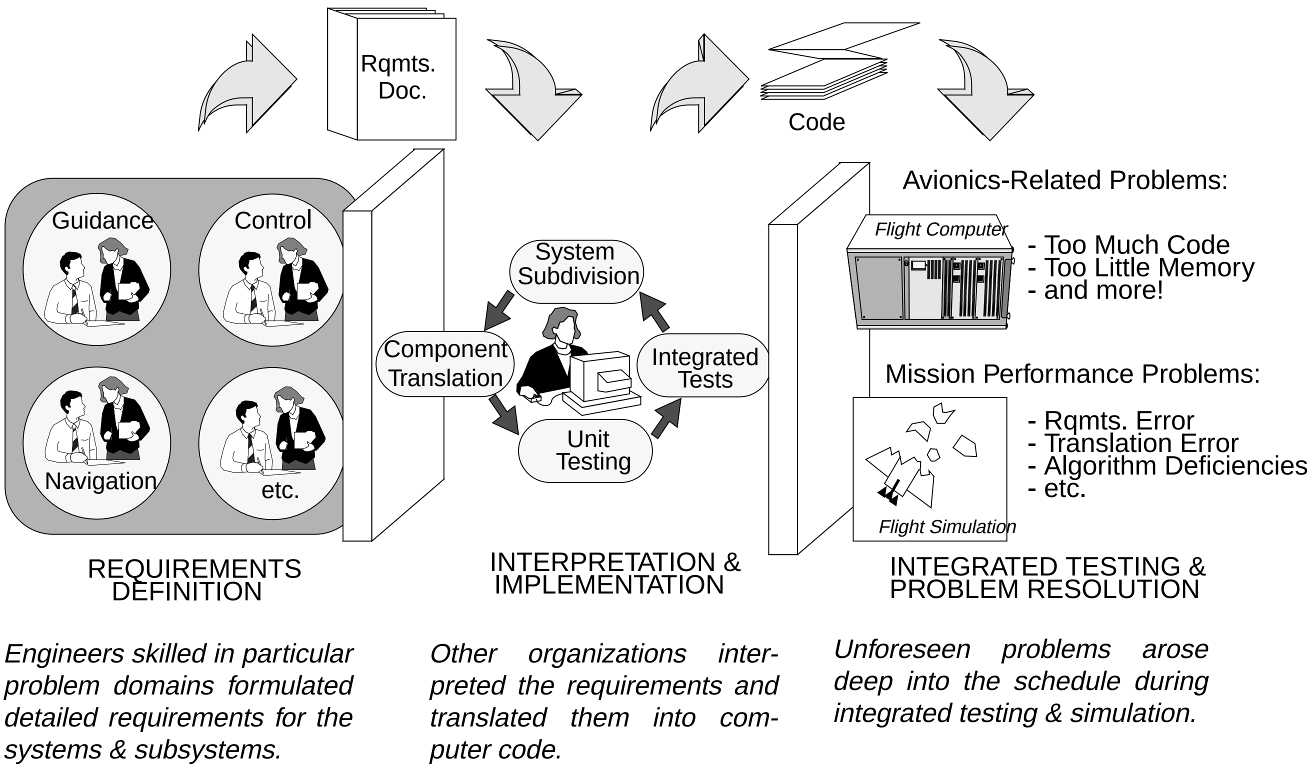


Figure 1. The Traditional Waterfall Flight Software Development Approach

2.2 Finding a better methodology for modern software development

While many system development efforts still claim to use the waterfall model, in the trenches programmers, analysts and project managers are devising more effective techniques. Today, these have to be forced into the waterfall life-cycle for external consumption; that is, to pass reviews, quality gates and sign-offs.

Is there a way to capture these more effective techniques and mold them into a life-cycle model that is effective in today's software engineering environment? Suggestions for doing just that are presented in this guidebook.

The project team incorporated practical experiences gained using techniques that facilitate the Rapid Development of high quality systems, especially in the context of GN&C flight systems. Using what we have learned, over time and with testing and validation, guidelines for Rapid Development and a system life-cycle model for Rapid Development were constructed.

This document encompasses several major topics relevant to Rapid Development of quality software systems. These include:

- Important issues, concepts and practical ideas that, based on the experience of the RDL staff and extensive research of the software community, support success in the Rapid Development of high quality systems, especially for GN&C applications.

- The phases of a proposed new life-cycle model, including the major topics of interest in each phase. This model is intended to be a formal systems engineering approach to modern software development.
- Project management issues using a Rapid Development approach. Classical development theory is rich with suggestions for managing the cost, quality, schedules and risk associated with software development. In adopting a new life-cycle, these issues must also be addressed.
- Processes that support software development and project management in a Rapid Development environment.
- Collecting and evaluating metrics in a Rapid Development environment. After studying the current state of the art of metrics data collection and evaluation, recommendations are presented for a start up metrics program in a Rapid Development environment. These include essential modifications to standard metrics so that they better support the new methodology.
- Tools that support and leverage Rapid Development methods. This is not an exhaustive review, but presents some current experiences.
- Lexicons of important Rapid Development and metrics terminology.
- References.

2.3 The Expected Payoff

Experience with projects that used some of the techniques of the Rapid Development methodology, indicates that high quality systems can be developed faster and with smaller, but more integrated teams, using this technique as opposed to the waterfall approach. Furthermore, experience shows that user satisfaction with the systems developed improves when using Rapid Development techniques. By defining and then applying a rigorous model that can consistently and dependably produce these results, it is anticipated that high quality systems can be developed with less risk, lower cost, and better adherence to schedules.

Some reasons for the observed success of this methodology include:

- Complete integrated systems are built early in the development cycle.
- Early integrated systems are often low in fidelity, with stubs for unavailable software or hardware components.
- “Systems” problems and interface problems are solved early.
- Due to the concurrent engineering approach, staffing requirements tend to be relatively level for much to the development effort.
- Prototype software and hardware systems are not thrown away, they evolve into the final product.
- Early integrated test builds customer and developer confidence.
- Milestones are determined with specific product focused objectives and acceptance criteria.
- Detailed development is done by Domain Experts.
- Integrated Rapid Development project teams are formed around the skills and expertise required to complete the project, including domain expertise, systems

expertise, and technical management

- The project team is responsible for:
  - integrating all project elements, and
  - Configuration Management and Quality Control, and
  - ensuring the project remains product focused.
- The project team takes ownership of the entire development process and end product.
  - End-to-end responsibility and ownership is more efficient and promotes a more productive work environment.

One important aspect of Rapid Development, the Spiral Development Process is an accelerated development process where the system requirements, design, code, test, and integrated test processes are iterated on concurrently rather than being executed sequentially and in a disjointed fashion. A spiral development process can make effective use of tools to integrate the requirements analysis, design, code and test (including test coverage) environments.

- An integrated environment involves the designer in all phases of product development.
- Changes anywhere in the requirements, design, or code are more easily implemented everywhere.
- CASE tools, including graphical user interface (GUI) simulation and modeling tools, coupled with autocode generation and real-time processor testing, are most effective in the hands of the integrated project team and speed the spiral development process.

The development tools available today and anticipated for the near future, such as CASE GUI simulation and modeling tools, are *not* just more advanced programming languages.

- Tools support integrated analysis, design coding and testing efforts.
- If an integrated toolset is not used in the requirements and design phase, then an additional step to translate logical flow into data-flow code is required.

3.0 Guidelines for Rapid Development

These are some of the most important features of the Rapid Development methodology that is successfully evolving in the GN&C RDL and as a result of this research:

3.1 Project Staff

Creation of a small integrated project team that is talented, knowledgeable in all key areas of the project, and able to follow the project for its entire life-cycle is critical. Most of the team members should be dedicated to the project full time especially after the initial concept development phase is completed. Rapid Development methods depend on rapid situation evaluation and response, and leveraging knowledge from one phase to the next. Construct the team carefully, and try to keep them together. If possible, once a team establishes a strong working relationship it should be maintained even across projects (although modified as needed to achieve the critical mix of expertise for each project). A working Rapid Development team is a valuable asset.

Involve the users and customers. Get commitments from them to be involved in solving the problems, reviewing the system, acceptance testing, etc. Make sure the project plan emphasizes the importance of this involvement. Part of what makes this approach rapid is getting things right as soon as possible to minimize misinterpretation, redesign, rewrites and change requests to already completed work. A close working relationship among the project team members and the users and customers helps make sure that the delivered system is the desired system

Users/Customers may exhibit marked differences in preferences and satisfaction levels. It is generally a mistake to assume that the developers will be able to negotiate a universally acceptable solution. Assign someone to have authority and responsibility for resolving conflicting requirements and desires.

Because the project team members are all intimately involved in the design and development, they may not maintain the objectivity necessary for full, independent, validation and verification. Plan to recruit knowledgeable sources or hold a periodic independent review outside the project team to assist with these efforts.

3.2 Tools to Support Rapid Development

A development approach should be chosen for each project based on the problem to be solved. The classic, or waterfall, development methodology is seldom effective for the types of problems we are trying to solve in the GN&C RDL. Specific recommendations for choosing among several different development approaches are discussed in a later section of this document.

Use advanced software engineering tools to greatly improve productivity. Investing in appropriate tools also adds to the flexibility and robustness of the system as well as the ability of the team to respond to problems and changes. Automated support for design, coding, testing, documentation, and configuration management are among the desirable options. The use of an automatic code generator, especially, mitigates the need to track and solve many problem areas, such as syntax errors, compiler idiosyncrasies, programming errors.

(However, the quality of auto-generated code needs special attention, due to the current state-of-the-art of this type of tool. This seems to be improving as the industry matures and more vendors are entering the market.)

Do not get locked in to a particular language, development tool, support tool or processor or try to force fit a tool where it does not belong. Choose the best tools for any particular task. Stay on the leading edge of the Rapid Development technology while it is still paying large productivity rewards, or check with others who may have recent experience with the tools being considered.

Plan for element reuse. Organize code to facilitate reuse. Invest in hardware, software and staff to create and maintain a reuse library. The library should support browsing capability, with searchable attributes, and include tailoring instructions for library elements. Investment upfront in reuse pays off long term. There are costs associated with establishing, populating and maintaining a reuse library. Management and budget support is crucial.

Implement automated release build capability to consistently interact with configuration management and build releases from known libraries.

3.3 Approaching the Problem

Work the system architecture, including interface requirements and dependencies and test and validation strategies, very early in the project life-cycle. Under classical development approaches, integration, communication, and interface issues have frequently been major problems. Under Rapid Development, it is recommended that end-to-end integration of the architecture begin early in the development cycle, so that many of these types of problems are resolved before the rest of the system has been developed. Dummy software stubs are usually sufficient to test the software and hardware connectivity in the early phases. Simulations may initially substitute for planned hardware modules. Retest the architecture as actual hardware and software become available. It is crucial to identify and validate the system architecture as early as possible.

Early documentation should emphasize functional requirements over implementation requirements. Design documentation should emphasize system architecture and interface requirements. Detail level for specifications should vary depending on risk level of the system element; defer detailed elaboration of the low risk elements until the high-risk elements of the design have stabilized.

Start tackling the hard problems first. These are high risk areas for a project. Use prototyping to test alternatives and choose the best (or reduce the set by eliminating alternatives found to be unworkable). As a project progresses, risk and uncertainty will be reduced by this approach. Some of these prototypes may be throw-aways. Once simple prototypes have helped find the correct way to solve a particular problem, go back and implement or evolve a production version (usually more robust and with more error checking and recovery modes than the prototype).

Perform hardware-in-the-loop testing earlier in the development cycle than has been historically common. As soon as actual flight or support hardware is available, and testing with it is practical, integrate it into the system, at least for testing. Again, the emphasis is on identifying potential problems as soon as possible and fixing them while it is still relatively easy to do so (i.e., without major reworks).

As much as possible, “as-built” documentation should be automatically generated by the software engineering tools. Documents should evolve with the system; add information to the appropriate documents when it is known and stable, delaying complete formal documentation of specific issues until after problems have been surfaced and successfully resolved.

3.4 Implementation Hints

Careful planning and a firm resolve are required to guard against “requirements creep” under a Rapid Development methodology model. This is the tendency to continue to demand additional functionality from a system until it has crept beyond the original scope of the project. This is a special risk in this type of development since one of the key characteristics of Rapid Development is the evolution of (detailed) requirements during the development phases. Clear traceability of requirements to distinguish “derived” requirements from “new” requirements is necessary. Set goals for each evolution cycle and for the overall project that will clearly identify when the project is finished. Consider the cost/benefit trade-offs whenever plans are modified.

Create and maintain automated test sequences for each system developed. Augment and rerun the test series to validate that system changes have not interfered adversely with existing capability. This applies to both development and maintenance cycles.

Plan for project turnover, from development to sustaining engineering, in parallel with design and development by involving the users and transition team. This can begin as soon as the design stabilizes.

Rapid Development does not imply ad hoc development. It is a fast paced, dynamic environment, typically with tight schedules and high expectations. Careful planning and monitoring is essential for success.

Many of the observations regarding the effectiveness of Rapid Development techniques have been learned and improved while working on small to medium sized projects. The concepts need additional testing on large projects, since issues of scale may well demand modifications to the techniques.

4.0 A Proposed Life-Cycle Model

In the long run, any systems development effort will have many common tasks, no matter what methodology is used. That is, requirements must be determined, code must be written, tested and validated, documentation must be written, and the project must be managed. There are various ways to order and perform these functions. The life-cycle model proposed in this document uses as its basis the model which has been successfully evolving in the GN&C RDL and augments it in ways that are designed to improve project management, software control, and verification and validation.

The proposed life-cycle model, from idea to obsolescence, including both system development and system maintenance, is comprised of the following major phases (also see Figure 2. Life-Cycle Major Phases on page 9):

- Project Initiation
- Project Evaluation
- Conceptualization
- Evolution
- Finalization
- Installation
- Sustaining Engineering
- Shutdown

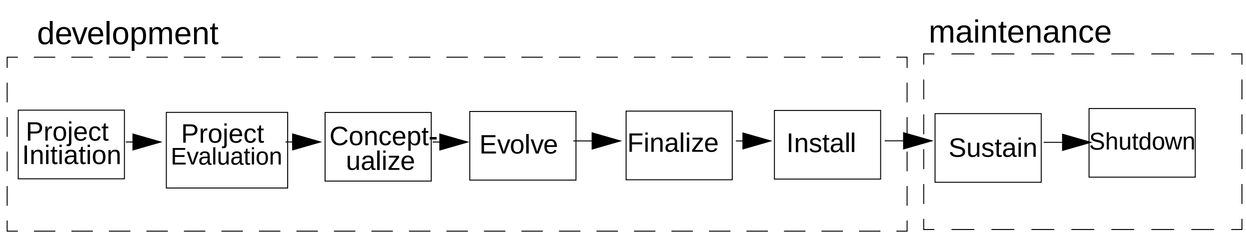


Figure 2. Life-Cycle Major Phases

The high level objectives of each phase are discussed below. A later section will address support processes for the model.

This document primarily addresses the development phases of the project life-cycle model. Under the Rapid Development paradigm, the guiding concept is “build a little, test a little, fly a little”. This approach tends to focus on design problems, technical issues, and implementation errors early in the development, before they propagate and while they are easier and cheaper to fix (relative to modifications made closer to or after delivery of a completed system). Using this approach, it is critical to maintain interaction with the target community (users and customers) throughout the development cycle. Similarly, documentation, project plans, schedules, and software releases are living entities under Rapid Development, to be revised and augmented as a project progresses, as more is learned about the problem to be solved, and as more details evolve and are implemented and validated. These critical interfaces are illustrated in Figure 3. Integrated Project Team Critical Interfaces on page 10.

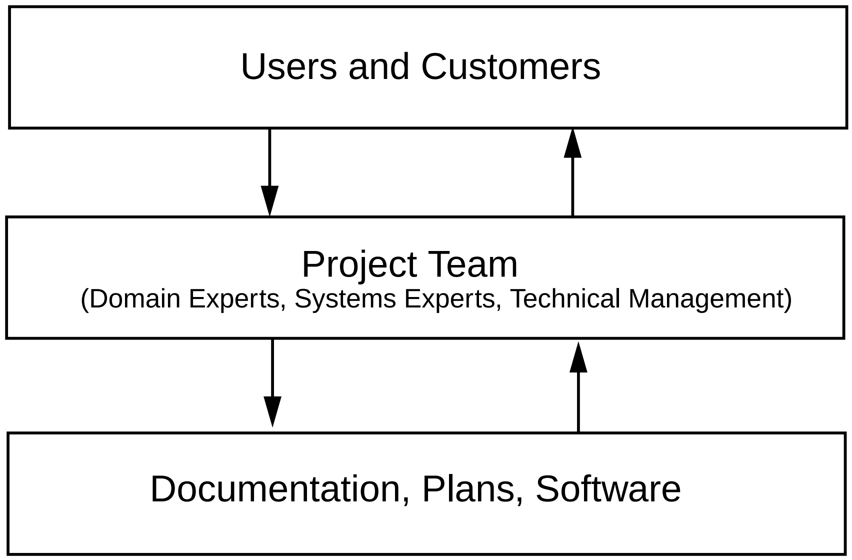


Figure 3. Integrated Project Team Critical Interfaces

5.0 The Development Phases of the Life-Cycle Model

5.1 High Level Objectives of Project Initiation Phase

The Project Initiation phase is the first step in determining new and potentially promising projects deserving of further study. The overall goal is to determine whether a project is needed, feasible (both in technical and budgetary terms), and compatible with the goals of the organization. Typically this phase will be user/customer initiated.

The purpose of this phase is to collect and present sufficient information about the problem and the proposed system, or system upgrade, to support a management decision about whether to proceed to the next project phase. This will include information about the skills required to perform the next phase.

5.1.1 The Proposal

The Initiation phase begins when someone identifies a problem and proposes to develop or upgrade a system to help solve it. The first step is to identify and state the problem to be solved.

Determine the high level functional requirements which must be met in order to solve the problem. In this phase, there is no need to define how the problem should be solved. Requirements need not be complete or detailed. Any hard requirements must be stated. Document what is known about the problem. Have customers complained? Are deadlines being missed? Is there a budget problem? Have changes occurred which require support in an entirely new area? In general, why must this problem be addressed?

State the known issues and identify areas of uncertainty. If there are known risks, identify them along with any assessments of the risk level and suggestions for managing the risk.

What will determine whether the problem has been solved? In general terms, state the success criteria from a business standpoint as well as technical and performance criteria.

Initial feasibility analysis is appropriate, but extensive analysis is not necessary at this phase. Just state what is known or an informed belief about the feasibility of the proposed project. Suggest alternative solution strategies. Identify the known relationships between the proposed project and other projects, including existing systems, systems under development, and others being proposed.

5.1.2 The Decision to Proceed

If the proposal is accepted, a decision is made to proceed to the Project Evaluation phase. This is not a commitment to the full project life-cycle. It is a commitment to further investigate the problem and the proposal.

To close out this phase, management should commit the necessary resources to the next phase. These include personnel, budget, equipment and training.

Personnel include a project advocate, or leader, who leads an integrated project team of systems experts who have or will be trained in the necessary skills and domain experts with expert understanding of the problem being addressed.

The next phase is not intended to be a full detailed requirements analysis. Management should clearly outline the budget, equipment and time frame which are being committed to the evaluation phase. Initial estimates of the overall project costs should also be made at this time. These will undergo refinement in succeeding project phases, if authorized.

It may also be appropriate to identify and pursue training and purchasing requirements, especially if they are needed for the next phase or if long lead times are involved and the project is likely to be funded. Are the training and purchase requirements consistent with maintaining state of the art capability in Rapid Development? If so, then making the investment now has the double payoff of improving overall capability while contributing to the momentum of the current project.

5.2 High Level Objectives of Project Evaluation

Here is where the project team really begins to look at how to solve the problem at hand. Working with the domain experts, the team will study in detail the problem to be solved. Rather than assign individuals to write various parts of the requirements, the integrated project team should initially meet as a group to exchange knowledge about the problem, discuss solution strategies, and consider options. The goals in this phase are to understand the problem and devise a solution strategy.

5.2.1 The Evaluation

The high level functional requirements will be produced in this phase, but they may look a bit different from traditional requirements. There may be areas in which two or more alternative sets of requirements are identified as possible solutions, or the requirement may be stated as a range. Some requirements may be desirable, others may be strictly required. Some areas may be unknown.

Part of this effort is feasibility analysis. The team should suggest and evaluate alternative solution strategies. These evaluations should consider (at least) cost, risk and probable outcome. Are there interdependencies with other projects, and if so how will that effect the feasibility of this effort? This may include technical dependency issues, but may also involve schedule dependencies and staff and resource sharing. Can these dependencies be exploited in ways that benefit this and other projects? At the conclusion of this phase, most alternatives should have been eliminated. It is, however, perfectly acceptable to enter into the implementation phases with some of these decisions still unmade, so long as the project plans include a strategy for further evaluation and decision making. The evaluation may include, for example, prototyping parts of the system in a variety of ways and applying selection criteria and tests to aid in decision making.

Another part of this phase is risk assessment. Identify the potential show stoppers, the really difficult parts of the problem. The difficulties may be technical, budgetary, time related, complexity related, or uncertainty about the problem or domain. Assess the level of risk. Compare various possible solutions. What is the impact of not doing this project, including risk to other projects or programs? Cost/benefit analysis may be useful. Plans for controlling risk should include early attention to the high risk areas, with reviews and decision points built into the schedule to reevaluate the risk potential. This will tend to contain the risks, by delaying implementation of the majority of the system until after the most challenging areas have been successfully designed. Similarly, in the project plans, detailed design of the more straight forward parts of the system should be delayed until after the design for the high risk areas has stabilized. The well defined, low risk, parts of the project should be relatively easy to design around the tough parts after they have been solved. The reverse strategy can be quite costly, and often involves extensive redesign and rewrites and can result in code that is more difficult to maintain.

Other issues that the project team should address in this phase include safety, security and privacy concerns, and initial estimates of the types and quantities of resources (staff, hardware, commercial software, other) required to implement the system.

When all of these issues are well understood, the team should choose a development strategy for the project. While this document is emphasizing a new Rapid Development methodology, each project should carefully consider whether this or another strategy is more appropriate for the problem being addressed.

For large projects, the team may wish to subdivide the effort into smaller efforts. In this case, the project partitioning should be defined, and a work breakdown structure should be defined. Project teams must be identified for each partition, and each team should proceed with development, beginning with this (Project Evaluation) phase. Plans for coordination and integration of the partitioned efforts must be specified.

5.2.2 Choosing a Development Strategy

Many of the ideas for improved software development methodology can be applied to any project. Yet each problem is different, and the project team needs to decide what is the best methodology to solve a particular problem. Tailor the guidelines for maximum success, and document the process to be used for each project as part of the project plan.

We distinguish among three primary approaches to software implementation: Waterfall, Incremental, and Evolutionary.

5.2.2.1 The Waterfall Model

The traditional Waterfall approach is characterized by distinct, sequential development phases, with separate hardware and software development paths and no integration until late in the life cycle. The exact number of phases and their definition may vary somewhat depending on project size and organization culture, but typically include the following:

- Requirements Definition
- Requirements Translation (System Design) and Review
- Software Design and Coding
- Software Test
- Integrated Test

The phases are often organized into “silos”, or distinct organizations often located far apart. Each technology discipline takes ownership of only a portion of the final product, and for only a certain phase of the program. Major milestones are well defined, and under government standards typically include:

- SRR: System Requirements Review:
- SDR: System Design Review
- SSR: Software Specification Review
- PDR: Preliminary Design Review
- CDR: Critical Design Review
- TRR: Test Readiness Review
- PCA, FCA: Physical and Functional Configuration Audits
- FQR: Formal Qualification Review

The classic Waterfall development model is best used when system requirements are straightforward, well understood and stable. The problem to be solved should be one that is well understood, with standard solutions. Funding should be stable and predictable.

5.2.2.2 The Incremental Model

The Incremental Model is characterized by a series of waterfall cycles that together complete a project. Usually only 2 or 3 cycles will be used for medium sized projects, often as many as 5 to 10 cycles for large complex systems. A system is delivered after each cycle with some subset of the final desired functionality but with each delivered function complete. For example, a timekeeping system might deliver the capability to support weekly operations in cycle one, periodic reporting functions in cycle two, and planning and forecasting functions in cycle three. This approach partitions the total problem to deliver some useful capability earlier than it would be possible to deliver the entire system. The incremental model is sometimes (mistakenly) used to claim that “Rapid Development” is being used within the standard government milestones.

An Incremental development strategy is recommended when the most critical functions required of the system are well understood and the project is not small. The system must lend itself to being divided into separate, complete, useful, stand-alone subsystems. These subsystems will usually be of varying levels of criticality, with the more critical functions implemented in earlier deliveries. The incremental approach helps ensure that highly critical functionality gets delivered as soon as possible, which is especially important under uncertain funding conditions. Pre-planned product improvement cycles are based on the Incremental Model.

5.2.2.3 The Evolutionary Model

In the Evolutionary Model, an integrated system is developed early and incrementally improved toward the final goal. The driving philosophy is “Build a little, Test a little, Fly a little.”

The evolutionary model resembles, but is not the same as, the original Spiral Development model as presented in the literature (especially Boehm, 1988). The original Spiral model is a type of evolutionary model in which the spirals are driven by a philosophy of risk reduction. Another similar approach presented in the literature relies primarily on rapid prototyping to discover system requirements before proceeding with implementation.

The Evolutionary Model builds on these approaches. A cyclic process is used to rapidly execute a development cycle. All activities (detailed requirements discovery, design, coding, testing) are essentially performed concurrently within a cycle. Requirements tend to evolve with the design. Each cycle has specific goals. The goals may be chosen, for example, to contain risk, achieve a desired level of fidelity, implement specific functions, or coordinate with other project schedules. The results of each cycle help determine the goals for succeeding cycles. Each cycle results in a complete, end-to-end, system. Hardware-in-the-loop testing is initiated earlier in the development cycle than with traditional methods. An integrated project team includes all the necessary skills and expertise and takes ownership of the entire process and end product.

In an Evolutionary Development process, the level of fidelity of the project increases over time until it is completed. The project is split into a series of logical milestones, or “Drops”. Each Drop represents an increased level of fidelity.

A Evolutionary Spiral Development methodology is recommended when:

- System requirements are vague or incomplete.
- The problem to be solved is new or not well understood. Solutions unknown, uncertain, or not obvious.
- Software development must occur concurrently with hardware development, contributing to the risk and uncertainty.

5.2.3 Rapid Prototyping

Any of these three development strategies, but especially the Evolutionary Development model, may be enhanced by the use of rapid prototyping techniques. That is, for those aspects of the system for which the best solution is not known, “quick and dirty” prototypes of alternatives can be employed to aid in decision making and drive the final design. These are typically not robust enough to be delivered, and are used to illuminate problems and alternatives and solidify design and implementation approach.

In many cases, these prototypes can be evolved to become the delivered code. In some cases, the prototypes are too rough, and it is more efficient to rewrite the prototype module as a robust system component. In either case, by using the same person or team to do both prototyping and final development, all of the knowledge gained in the prototyping effort is used in final implementation, and this continuity tends to improve both the speed and quality of implementation.

5.2.4 Hybrid Approaches

In many cases, some hybrid approach may be preferred. This is especially true for relatively large projects where the various project partitions may be developed using different approaches, or the high level project management may proceed differently from detailed subsystem project management.

5.2.5 The GN&C RDL Preferred Model

In the GN&C RDL, an Evolutionary Spiral Development methodology enhanced by Rapid Prototyping techniques has been successfully applied. This approach will be assumed for the remainder of this document (see Figure 4. Evolutionary Development Life-Cycle Model on page 16).

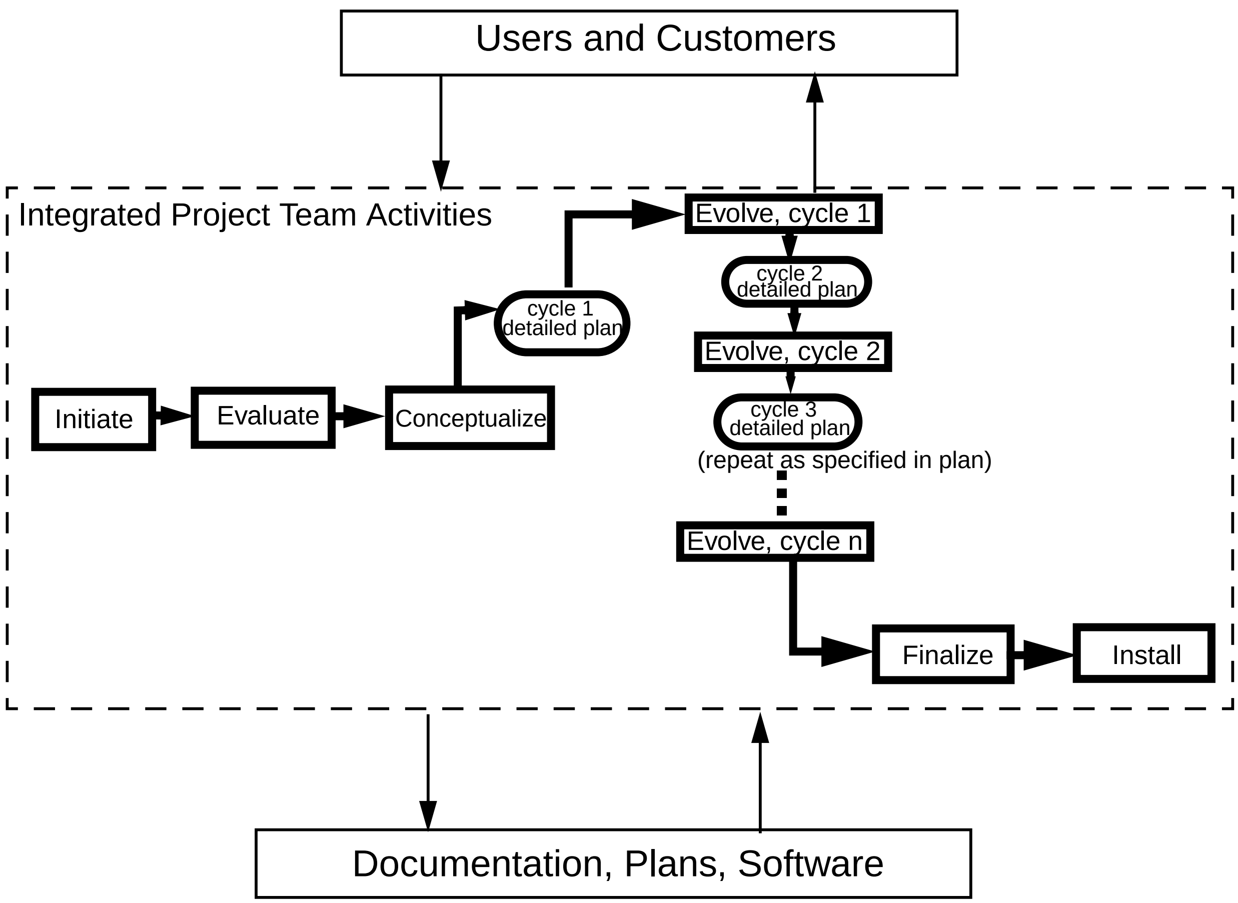


Figure 4. Evolutionary Development Life-Cycle Model

5.3 High Level Objectives of Conceptualization Using an Evolutionary Spiral Development Life-Cycle

Entering this phase, the project team has a good understanding of the problem to be solved, the risks involved, possible solution strategies, and has chosen a development approach.

In the conceptualization phase, the team prepares for implementation. The software engineering environment that will be used for implementation, including tools, facilities, hardware, processes and procedures, should be defined, procured if necessary, and installed.

The primary deliverables of this phase are the system level functional requirements, the high level system architecture, and the project implementation plans. If applicable, an analysis of dependencies of this project on other projects and systems may also be produced in this phase.

5.3.1 The Functional Requirements

The functional requirements document should be fairly detailed as it will drive the system implementation in later phases. Note that no detailed requirements or detailed design document is completed prior to implementation. In our evolutionary Rapid Development environment, implementation and design details will be discovered as the system is implemented. It is therefore imperative that the functional requirements document be a clear, solid and complete description of the required functionality of the system.

5.3.2 The System Architecture

High level system architecture may be specified by diagrams, but whenever possible it is also desirable to implement a working prototype of the architecture design. This can be an extremely low fidelity implementation, with most functional modules stubbed out, if necessary. By implementing an integrated end-to-end prototype early in development of the system, many interface issues can be solved before the system has been coded. Experience has shown that these problems are easier and cheaper to fix early in the implementation of a system, when less code will have to be rewritten to accommodate new interfaces. To the extent possible, it is desirable to perform hardware-in-the-loop testing on this early system prototype. Here again, the goal is to identify and fix potential interface problems early in the development of the system.

After any problems that surfaced have been solved, the resulting prototype (often called the Phase Zero implementation) serves as the initial high level system design. Note that the design is implicit in the successful implementation, rather than design driving the implementation. This gives implementors flexibility to evolve the best design that both works and fits the functional requirements. “As-built” design documents should be produced, and should evolve with the system’s implementation, but it should be possible to automate much of the work to prepare such documentation.

5.3.3 System Dependency Analysis

In some cases, a system dependency analysis may be useful.

The dependency analysis should include functional and data dependencies between this and other systems, existing or planned. Any assumptions made for this system that imply levying requirements on other systems should be called out, and such information should be communicated to the appropriate project teams.

Dependency analysis should also include reusability analysis. Look in the reuse library for existing code that can be used on the current project, or can be easily modified for use on the current project. Also, identify areas where work done for this project could benefit other projects.

5.3.4 The Project Implementation Plans

Two levels of project implementation plans should be produced. The overall implementation plan will show plans for the remainder of the development phases of the system life-cycle at a moderate level of detail. A more detailed plan will be prepared for the cycle which immediately follows this one, Cycle One of System Evolution.

5.3.4.1 Implementation Plan for remainder of project development

The overall plan for implementation of the system should include:

- Goals
- Scope of Effort
- User/Customer responsibilities
- Deliverables
- Number of Evolution cycles (Build strategy)
- System level objectives of project and of each cycle
- Preliminary Project Schedule
- Cost Estimates
- Procurement plan
- Verification/validation requirements
    - (including Test approach, strategy, and requirements)
- Configuration management plans
- Documentation requirements

The implementation plan is not a static document. As implementation progresses, it should be updated (at least at the end of each evolutionary cycle) to reflect current knowledge of the project. In general, certainty should increase with each cycle and plan update.

5.3.4.2 Detailed Plan for cycle one of System Evolution

Initially, the detailed plan for evolution cycle one should be prepared. Then, for each evolution cycle, one of the exit conditions of the cycle is the completion of a detailed plan for the next cycle. Lessons learned in the cycle and implications of the detailed plan for the next cycle may impact the overall implementation plan, in which case it should be updated as well.

Topics which should be considered, and included as relevant, for the cycle detail plan include:

- objectives
- constraints
- alternatives
- risk areas
- schedule
- cost estimates
- planned deliveries, including documentation and updates
- Verification/validation plans
    - test strategy, including HIL, OIL, and end-to-end integration tests

- configuration management issues

5.4 High Level Objectives of System Evolution Phase

This is the phase where the majority of system implementation takes place. The primary goal is to evolve the system according to Overall Plan developed in the Conceptualization Phase.

The evolutionary phase will typically be divided into several sub-phases, called “cycles”. The number of cycles planned and the major system level objectives for each cycle will have been defined during the Conceptualization phase. As a rule of thumb, objectives should get more concrete with each succeeding cycle.

Typically, the plan for a cycle will call for maturing some subset of the system functions to specified levels. It is often a good development strategy to concentrate on the more difficult, less understood, more risky modules first, as prototypes. This way, each cycle reduces uncertainty in the project and its budget and schedule.

Each cycle should include user/customer evaluation and documented feedback. The following cycle should address this feedback. This increases the likelihood of achieving a high level of user and customer satisfaction with the final product.

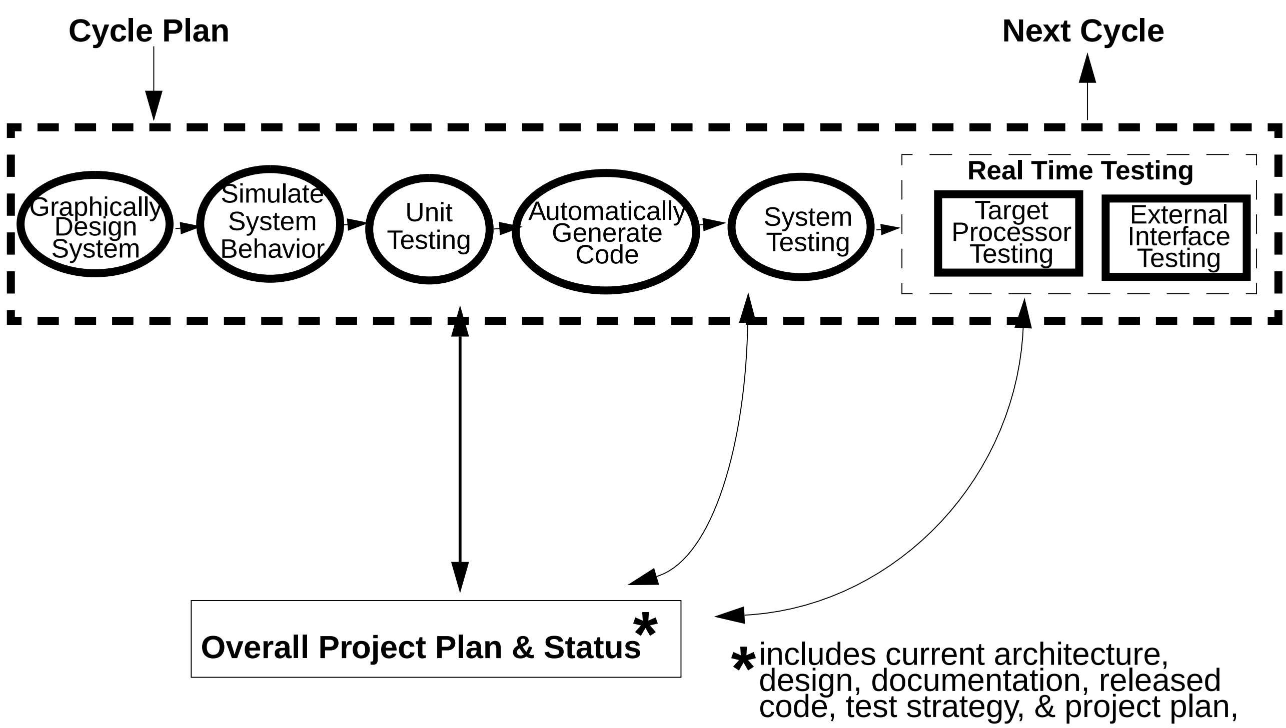


Figure 5. One Cycle in the Evolution Phase: Additional Detail

Lessons learned in any cycle may lead the team to revise the overall plan developed in the previous (Conceptualization) phase. In this case, it is important to focus on the project goals and carefully evaluate the benefits of the changes versus the costs of not making the changes. Find the correct level of change for success while keeping the project on track and avoiding “requirements creep”.

5.4.1 Objectives of Each Evolutionary Cycle

The principal objective of each cycle of the evolution phase is to complete the interim products and deliverables that meet the planned goals for that cycle.

Deliverables for each cycle include software, test cases, and documentation. All should be placed under configuration management. To complete a cycle, the software delivered should have completed unit testing, integration testing and validation and evaluation by users or customers as appropriate. Documentation produced in previous phases or evolutionary cycles should be updated to show all revisions and additions. This will include at a minimum the functional requirements, system architecture and overall project plan. As the system evolves, design and implementation details should also be captured in as-built system design documentation. Other documents, products and deliverables may be required, as called for in the overall implementation plan and the detailed plan for the cycle.

Each cycle should conclude with a report which details the results of that cycle. The report should specifically address the planned objectives of that cycle. Were the objectives of the cycle met? How, or why not? What alternatives are available for missed objectives? The report should include response to user input from the previous cycle. This could include design or implementation changes, cost/benefit trade-offs, actions taken, results, or other responses. A user evaluation for this cycle should also be included.

Depending on the plan for the cycle and the results achieved, other information may be appropriate to include in the cycle report. It could address design constraints that were included, along with rationale. Where alternatives were previously identified, the report should indicate which alternative was selected, the selection criteria used, and the implications of the decision on this and other systems.

Other issues which need to be addressed, as appropriate, include:

- risk resolution/results
- schedule impacts, modifications
- cost
- deliveries
- test results

Before completing the cycle, the project team should reassess feasibility in light of results achieved in this cycle. A detailed plan should be prepared for the following cycle. Lessons learned in this cycle, as well as plans for the next cycle, may impact the overall implementation plan. If this is the case, then the implementation plan must be updated as well. Update other documentation, such as system architecture and as-built design documents, as appropriate.

Migrate new work to reuse library, as appropriate.

5.5 High Level Objectives of Finalization Phase

This phase is really just the last planned evolution cycle, but there should not be any remaining issues when this cycle completes. Note that when using evolutionary and prototyping techniques to speed system development, performance tuning and stress testing often are quite critical activities at this point in the development.

Primary elements of this phase include:

- Performance tuning
- Stress testing
- Finalize documentation
- IV&V
- User, customer and developer sign-offs
- Establish plans for user support, maintenance and upgrades
- Installation and Transition plans

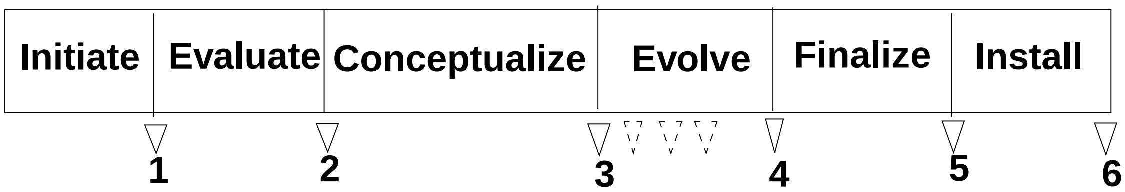
5.6 High Level Objectives of Installation Phase

In the Installation phase, the system is made available for its intended use. Activities include setting up scripts and procedures for everyday use of the system. User support, maintenance and upgrade plans, from the previous phase, should be initiated. Support for user training and start-up activities is required.

Since at least some of the user community have been involved in the development process, the Rapid Development methodology should facilitate smooth installation of the completed system.

5.7 Summary

The figure below restates, in summary fashion, the major phases of development under the proposed Rapid Development life-cycle model. The previous sections have discussed primary activities of each phase. Some of the most important of these are called out again in the figure. To support these activities, key software engineering support processes will be outlined in a later section.



1: Statement of problem to be solved; Decision (continue project or not)*; if project to be continued: Composition of design team; Commit Budget for next phase.

2: Feasibility & Risk Assessment report; Development Strategy selection

3: Overall project plan, including reuse, software engineering tools, hardware, build strategy; Detailed plan for cycle 1; High level system architecture prototype;

4: After each cycle: Status wrt current cycle plan and overall project plan; Detailed plan for next cycle; Updated project plan

5: Final Documentation; Sign-offs; Installation & Transition plans; Maintenance & support plans

6: Installation completed; Start-up completed; Training completed

* This decision point is implied for each phase, even though not explicitly stated.

Figure 6. Project Management Milestones

6.0 The Maintenance Phases of the Life-Cycle Model

In the most part, the maintenance phases of this lifecycle model will be quite similar to traditional methods. They will differ primarily in the emphasis on life expectancy evaluation (for long term planning), in the Sustaining Engineering phase, and reuse consideration in the Shutdown phase.

6.1 High Level Objectives of Sustaining Engineering

The principal objective of Sustaining Engineering is to maintain system usefulness by protecting system integrity, fixing problems that are identified, and performing modifications to keep up with changing environments. Documentation management, including updates and configuration control, are part of this effort.

Several secondary activities support the primary objective. These include supporting user activities, providing user help support, and user training.

On a regular basis, the system should be evaluated to ascertain remaining life expectancy of the system. The evaluation should include some analysis of the cost of maintaining the existing system versus the cost of replacement. If the need to replace or significantly upgrade the system is anticipated, then it is desirable to include an estimate of lead time required and potential cost. Keep a running list of prioritized documented potential upgrades to use as input to any upgrade projects. Plan and lobby for replacement, if needed.

6.2 High Level Objectives of Shutdown

If the system is determined to have reached the end of its useful life-cycle, and if any necessary replacements have been installed, then an orderly shutdown is called for. This could include:

- Verify that all relevant elements have been migrated to the reuse library
- Archive software, documentation and hardware
- Release licenses
- Surplus hardware
- Assist users with migration to new system and procedures

7.0 Key Support Processes for Rapid Development

In many ways, the key support processes for Rapid Development are similar to those used in traditional development paradigms. This section will therefore briefly introduce the concepts and highlight some suggested modifications for Rapid Development.

7.1 The Need for and Application of Support Processes

Support processes are intended to control the development process in ways that improve chances for success. That is, the support processes are put in place to ensure that the system is developed correctly, on time, of adequate quality, and on budget, that the documentation is complete and the code is safely stored and retrievable, and that management is informed and aware of progress, problems and results of a project.

Support processes complement the life cycle, provide feedback to management on the progress of development, and provide information which can be used to drive process improvement.

As with the life cycle, the particular support processes used and their implementation should be customized specifically to best support each particular project. Moreover, the customization process should take into account the skills and experiences of the project team, taking advantage of any history and expertise with specific products, techniques or processes.

In choosing and implementing support processes for a project, the following issues and questions should be considered:

- How will the technical process be controlled?
- How will the use of resources (staff, budget, equipment) be budgeted, tracked, and controlled?
- How will project planning be done, both initially and in response to actual progress and status?
- What are the key risk areas for the project? How will they be identified, tracked and controlled?
- What are the key data products of the project? How will the data products and software products be managed and controlled?
- How will document content be managed and controlled?
- What tools and equipment will be used for this project? Include tools and equipment used both for development and for support processes.
- How can the system development process be measured to identify quality, cost, and schedule status and issues?

The answers to these questions will help determine what types of support processes are required for the project and how they should be implemented.

7.2 Types of Support Processes

One way to characterize system development processes is as either technical, management or institutional processes. In this view, the technical processes support completion of the tasks needed to perform a project, management processes support the tasks needed to monitor and control progress and resources for a project, and institutional processes support the tasks needed to maintain the organization and environment in which a project takes place. The following sections will explore each of these process types in more detail and discuss key issues and questions to consider when setting up support process tools and procedures for a project.

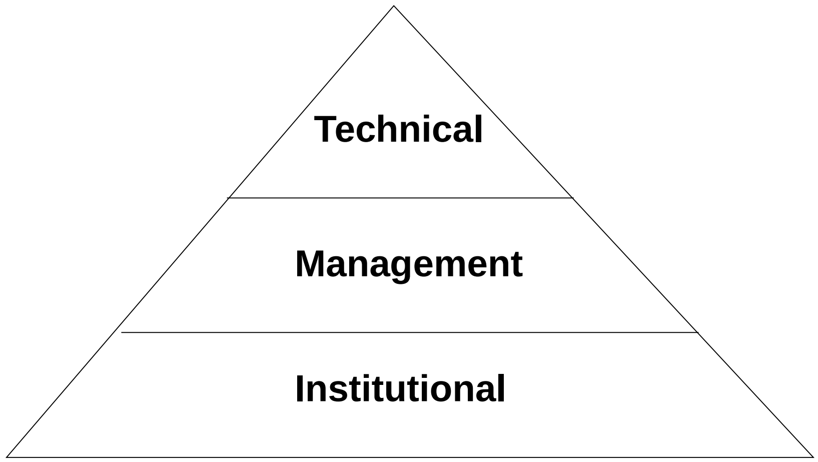


Figure 7. The Support Processes Pyramid

7.2.1 Technical Processes

Technical processes produce the product. These include all the steps in a project life cycle (discussed in some detail in previous sections of this guidebook), along with test and V&V (verification and validation) processes.

Management of the technical process includes monitoring technical issues with respect to the project plan. Experience shows that the exact method for doing this is highly individual, depending on the style of a particular project leader and that of the team members.

There are some key ideas to keep in mind when doing technical management in a Rapid Development environment. To achieve maximum success, flexibility and responsiveness must dominate project tracking. A strong project leader should as much as possible anticipate problems and have alternatives identified. Team members should quickly inform the project leader of any difficulties with potential schedule impacts. Frequent replanning will usually be required. Technical management of Rapid Development projects is a high energy, highly interactive process. To be most effective, the project leader and team members should have considerable authority to revise, rework, and reassign tasks, priorities and resources as needed to meet deadlines, budgets and requirements.

Team communication is a key element for success. All team members need access to the latest plans, schedules, requirements, priorities and decisions. There are many ways to achieve this, from a centrally located notebook that is updated frequently, to on-line web pages accessible by the whole team. Frequent status tag-ups can be useful, but are not a substitute for written material.

Frequent technical status meetings can be quite useful if they are short and focused. Strong technical management should guide these meetings to assure that needed information is exchanged but details not of interest to the entire team are worked independently. Avoid the trap of over discussing issues by having a clear understanding of who has decision making authority; it is not necessary to achieve consensus in every area. It will generally improve the project to solicit team input for central issues or especially difficult design areas, but too much discussion can impede progress.

Problems which may result in significant modification of the plan, whether in delivery content, milestone dates, or resource usage, should be communicated to management in a timely manner. Keep in mind that management abhors surprises, and can support the team better if status and needs are clear and up to date.

7.2.2 Management Processes

Management support processes are put in place to monitor and measure progress, while providing resources and support to the technical project team.

Key management processes generally include Resource Management, Project Management, Risk Management, Configuration Management, Test Management, Data & Documentation Management, Problem Reporting and Resolution, and Information Resource Management.

7.2.2.1 Resource Management

Resource Management includes tools and procedures used to budget, track and control the resources that are available to projects in an organization. Typically these resources include staff, budget and equipment. The primary issues to be addressed by resource management are:

- What resources does the project require? For each required resource,
  - When in the project life cycle is the resource required?
  - Must the resource be dedicated to the project or can it be shared?
  - How does the commitment required of the resource vary as the project progresses?
- What resources are available to the project to fill these requirements?
  - Are all the required capabilities represented in sufficient quantity?
- How does resource availability correlate to resource requirements during the project life cycle
  - There may be a need to coordinate with other projects.
  - This will affect project schedule.

Like most aspects of any project, and especially those using Rapid Development techniques, the resource requirements profile is likely to vary from the plan somewhat as the project progresses. It is important to track the variations and project their implications to ensure that resource allocation plans are still feasible. It is usually counterproductive to insist on exact balances in the resource allocation, since requirements are based on estimates which will vary somewhat from actuals anyway. It is more useful and important to plan for approximate balance, and then monitor and revise plans frequently as necessary.

Effective resource management in a Rapid Development environment can be challenging. A Rapid Development approach to systems implementation may require more detailed resource management than traditional models. This is because skill requirements may overlap among more phases of the system life cycle than the more partitioned classical approaches.

7.2.2.2 Project Management

Project Management includes the tools and procedures used to plan, schedule, and monitor project progress.

Project planning involves identifying the tasks that need to be done, and any interdependencies. Estimate the resources (time, staff, equipment, budget) required to perform the task. Determine significant milestones, deliverables, and any entry or exit conditions for the tasks.

The project schedule assigns resources to each task and lays the tasks and milestones out on a timeline.

Project management procedures and project progress will determine how often actual progress will be compared to planned progress and how and when the plan and schedule will be updated.

There are many things that cause actual progress to deviate from plans and schedules. A task may prove to be harder or easier than expected. Resources may not be available as expected, due to procurement difficulties or unexpected requirements from other projects. Internal or external ICD (Interface Control Document) conditions may not be on schedule. New tasks may be identified that are required but were not previously in the plan. The general expectation is that the near term portion of schedules will be more accurate than the long term portions of schedules.

In the Rapid Development environment using an evolutionary/spiral implementation model, an overall project plan and preliminary schedule should be prepared as part of the Conceptualization Phase. This high level plan will include an outline of the planned evolutionary cycles and goals for each, including entry and exit conditions. At the same time, detailed schedules for the first evolutionary cycle must be prepared. Since the detailed plans for each succeeding cycle will depend somewhat on the lessons learned in the current cycle, detailed scheduling will often be deferred for a cycle until near the end of the previous cycle. Yet a best estimate look at plans and schedules for the entire project is highly beneficial, in order to obtain and commit the necessary resources, to know when the system will be available, and to know when the team and other resources can be used for other projects.

7.2.2.3 Risk Management

Risk Management processes are used to assess, control and minimize project risk. Risk areas for a particular project could include cost, schedule, quality, safety, security and feasibility.

The nature of the Rapid Development life cycle introduces some special risk areas for a development effort. This is primarily due to the evolutionary nature of system requirements and project plans in this environment. Some of the special issues to be aware of include:

- Requirements Creep. As the life cycle progresses, more becomes known about the problem and its solution. When using Rapid Development, requirements discovery is a natural part of each development phase. As users and customers begin to see and use the results, the project team will naturally think of additional capability and new ways to use the system that are desirable but may be outside the original scope of the project. Since each cycle and phase can legitimately result in modifications to the requirements, there is a risk of constantly increasing the scope of the project. This, of course, can result in missed deliveries and cost overruns. Yet refusing to respond to changing requirements can negate some of the benefit derived from the Rapid Development environment. To solve this dilemma, the team must stay focused on the scope of the project. New or modified requirements requests should include an indicator of their criticality and estimated costs. When changes to the requirements are likely to result in changes in the agreed costs or schedule, management and customer agreement should be obtained. In formal environments, such as support contractors, this agreement should be in writing and should include the relevant contracts personnel. Sometimes desirable capability will not be implemented or will be deferred to a later project. Try not to let this reduce the dynamics and enthusiasm of the team; identifying and documenting ideas for future implementation is valuable, and delivering a good system on time and in budget for the current effort increases the chances that additional work will be funded later.
- Inconsistent expectations. The nature of the Rapid Development environment is that things change quickly and agreements tend to come in meetings or brainstorming sessions. But we all know that different people can come away from the same meeting with quite different ideas about what was agreed upon. Take the time to summarize and distribute updates with key points, and verify agreement among interested parties, no matter how great the temptation to “just do it” may be. Rapid Development techniques may skip some of the traditional upfront documentation and review steps, but still requires that major requirements and design decisions be recorded and approved as they evolve with the system.

Some aspects of the Rapid Development model tend to reduce project risk. Typically, plans for the evolutionary cycles will identify the highest risk areas of the project and work those first. After the solutions to the hardest problems have been determined, designed, and implemented or prototyped, the lower risk, easier, better understood problem areas can be addressed. It should be relatively straightforward to adapt the low risk system functions to the (then) in place solutions to the high risk problems.

The Rapid Development life cycle also encourages risk containment by:

- limiting the work that must be done before system development begins
- involving users and customers in development decisions, to improve user acceptance of systems
- frequent replanning improves flexibility to respond to changing requirements and budgets
- interim cycle deliveries contrast with the “all or nothing” mentality of traditional development and, in the face of budget uncertainty, ensure that completed work is

captured in usable form at predictable points in the project

7.2.2.4 Configuration Management

Configuration management systems and procedures (CM) define, implement and enforce the ability to track project information.

The most common application is software tracking. The configuration management process maintains the official software and tracks changes to it.

A complete configuration management system will usually include additional information. Common elements include documentation, test cases and results, and system support specifications (such as compiler requirements, library dependencies, and hardware and operating system configuration needed to support the release).

There are some special considerations for configuration management in a Rapid Development environment.

Usually there will be several official releases of the system (at cycle intervals). At least for the duration of the development phases, configuration management should track and support each release. This means freezing a snapshot at release time, and maintaining access to that snapshot, and previous snapshots, while development of succeeding releases progress.

Each official release should include documentation that also corresponds to that release, such as requirements and design documents. Capturing the detailed plan for the cycle is also recommended, since this will give information about the goals and capabilities of that release.

Test data, test drivers and test results are important elements to capture and place under configuration management. Also, system configuration requirements (hardware, operating system, compiler used, etc.) should be recorded for each release.

Whether there is a need to maintain interim versions at the conclusion of the development effort is project dependent, and the CM should include decision checkpoints for this.

The configuration management process must include mechanisms and controls for evolving between cycles. A variety of COTS tools are available to support check-out, modification, and check-in of files.

7.2.2.5 Test Management

The Test Management process defines required testing procedures, the process for ensuring and documenting that all required tests have been performed successfully, and the conditions that require repeating some or all of any test series.

When working in a Rapid Development environment, several deliveries (cycle drops) are usually planned for a project. Complete, independent verification and validation (IV&V) is often done only in the finalization phase. Unit testing, subsystem testing and integration testing is needed for each evolutionary cycle. At the completion of each development cycle, sufficient testing should have been completed so that the team is confident that the release works correctly or has documented known problems and included problem disposition in the project and cycle plans.

Specific test requirements should be included in the detail plan for each cycle. It may be necessary to repeat some tests from previous cycles to be sure that the current cycle's updates have not inadvertently caused problems in previously completed work. Other test runs from previous cycles may no longer be appropriate or may require modification to reflect the current state of the system.

As much as possible, the test process should be automated, to save time and to simplify repetitive and repeatable testing. Coordinate with Configuration Management to archive test cases, drivers, results, and interpretation.

After development has completed and the system has been delivered, it may not be necessary to maintain interim cycle test information. This issue is highly project dependent. Consider whether any of the intermediate cycle deliveries might serve as starting points for related or future projects. If so, the entire test suite may be valuable. Will any modules migrate to the software reuse library? In this case, certainly unit test data and drivers will be useful.

7.2.2.6 Data & Document Management

Data and Document Management processes define data and document requirements for a project, as well as responsibility for their creation, procedures for approval and distribution, procedures for archiving, maintaining and updating project data and documentation.

A Rapid Development environment creates some non-traditional issues in this area, especially documentation. Specific documents to be written, and their timing with respect to the project life cycle are often markedly different from more traditional models. Subsystem design documentation usually follows or parallels implementation. Requirements documentation tends to be more functional than detailed and may evolve with the system. The implementation generally serves as the detailed design, especially if graphic development tools (with auto-coders) are used. Updated documents may be released with each implementation cycle.

Released documentation should be held under configuration management. Until project development is completed (and, presumably, documents are in their final form), it is probably best to limit distribution of the documents. Accessible electronic copies of documents are recommended, since they allow access as needed to the most current versions. Excessive distribution of interim documents runs the risk of confusing or overwhelming recipients who receive copies of several versions. Interim versions should be clearly marked as such, to minimize confusion.

If some form of electronic approval capability (equivalent to signatures on a hard copy) could be implemented, it would be possible to also maintain the “official” copies electronically.

Whether or not to maintain availability of interim versions should be decided on a per project basis.

7.2.2.7 Problem Reporting and Resolution

Problem Reporting and Resolution processes define and carry out the formal mechanism for logging observed problems, facilitating their resolution and tracking their status.

Problem Reporting and Resolution processes are used during testing and during the sustaining engineering phases of a system life cycle. Traditionally, reported problems have been categorized as

- discrepancy reports(dr): documents an aspect of the system which does not match stated requirements
- change requests (cr): documents a need to modify one or more stated requirements and the system
- trouble reports(tr): documents a general problem in operating the system

In the Rapid Development environment, during system development phases, requirements are expected to evolve with the system, so that one end product of each evolutionary phase is updates to the system requirements. These may be additional requirements, or clarification of general requirements previously outlined in the requirements documentation. As these requirements are identified, they need to be logged and tracked as well. These may be included in the problem reporting system as a fourth category, requirements change.

- requirements change (rc): documents a newly discovered requirement or a clarification of a general requirement

Note the difference between a cr, which documents a change to existing requirements, and rc, which documents a newly discovered requirement.

Disposition of requirements change reports must be carefully managed to avoid the problems of requirements creep (see section 7.2.2.3, Risk Management).

7.2.2.8 IRM (Information Resource Management)

Information Resource Management deals with issues pertaining to tools, systems and equipment that will be used for a project. These could, for example, be data base tools, compilers, CASE tools, development environment, computers, networks, operating systems, and office support tools such as word processors and spreadsheets programs. An information resource management plan works to ensure that the necessary resources are available to support the project. For small projects, the plan is often unwritten and relatively informal. More complex projects may require more formality in managing these resources. In any case, it is important to investigate and plan for the availability and stability of the information resources that are required by the project.

Project plans and schedules can be effected by the information resource management plan in a variety of ways. If there are a limited number of licenses or workstations, there may be competition among team members or between teams for those resources. If a piece of software or hardware undergoes an upgrade there may be conversion requirements or down time interruptions of the schedule. There may be a need for technical support, both internal (from your organization) and external (usually the product's vendor), and the project budget must account for this. New or upgraded tools may necessitate training time and costs, or may have a "learning curve" effect, temporarily reducing productivity. System Administrators may be willing to accelerate or delay installations or upgrades to facilitate a project, or there may be conflicting requirements among projects that need to be negotiated.

Of special interest in the Rapid Development is the effect of the evolution of the information resource environment and its effect on intermediate deliveries. For example, if a compiler is upgraded, will there be a need to test interim releases of the system, or only the version currently evolving? Or, what happens to electronic project records (documentation, meeting notes, etc.) if the document preparation package changes? The answers to such questions are, of course, project dependent. It is important to be certain that they are considered.

7.2.3 Institutional Processes

Institutional processes are not project dependent, but rather project supportive. That is, these are processes which are put in place to generally improve the ability of an organization to develop good, cost effective systems in a timely manner. Some important examples of institutional processes include Labor Accounting, Process Improvement, Staff Training, Tool Evaluation & Selection, and Metrics Data Collection, Evaluation and Reporting.

7.2.3.1 Labor Accounting

A labor accounting process provides the ability to measure and track labor costs. Labor accounting may be done at varying granularity (minutes, hours, days), and detail levels (project, phase, task, department) depending on the needs of management.

Some examples of the uses for labor accounting data include:

- to monitor progress by showing what is being worked on
- to compare estimated to actual effort required to complete a task
- as historical data to improve future estimation capability
- to determine project costs, for internal accounting or customer billing
- to determine productivity trends and other project management metrics

7.2.3.2 Process Improvement

How does an organization know if its processes and procedures are effective and efficient? Process Improvement processes examine current ways of doing business, identify areas of weakness or potential improvement, then propose, implement and evaluate new methods as appropriate.

ISO-9000 and the SEI (Software Engineering Institute) CMM (Capability Maturity Model) are two approaches to process improvement that are widely accepted.

The Rapid Development guidelines in this document are a major effort at overall process improvement.

7.2.3.3 Training

In addition to project specific training, there is a general organizational need to keep staff up to date on current tools, processes, technology, etc.

When an organization starts using the Rapid Development model, project staff may not be familiar with Rapid Development the techniques and tools. Start up training will be necessary.

7.2.3.4 Tool & Equipment Evaluation & Selection

What is the best way to equip a work area? Tool and Equipment Evaluation and Selection is an ongoing, continuous process, because the state of the art advances, new products are offered, current products are upgraded, equipment wears out. Planning and budgeting for this activity will help ensure its success. If possible, survey staff for product or capability wish lists, and then keep a lookout for them in the marketplace. Document product evaluations for use by others and comparison with other products. If possible, share evaluations with other organizations, to get alternative views and to increase the possible scope of the evaluations.

Tools which support Rapid Development proliferate. To keep on the cutting edge, active attention to their evaluation is recommended.

As selection and procurement are planned and carried out, coordinate with training efforts. Consider effects on project plans, schedules and budgets (see section 7.2.2.8, Information Resource Management)

7.2.3.5 Metrics Data Collection, Evaluation and Reporting

Metrics related processes pertain to the collection, evaluation and reporting of project data. Metrics can be used to support project development, maintenance and management functions, as well as process improvement functions. It is an Institutional process because of the advantage to an organization of using consistent tools and procedures for metrics across projects. A consistent approach allows for project to project comparisons and improves the data collection results, as staff become familiar with what is expected and adopt it as part of the normal work environment.

Expect an effective metrics program to take two to four years to mature. Look for balance in the amount of data collected versus the effort needed to collect it. Automate the collection and reporting process as much as possible.

Evaluation of metrics can sometimes lead to unexpected results. Be open to new possibilities. Expect some initial staff resistance to the concept; acceptance will likely follow once the value of the program is demonstrated.

When choosing what data to collect and how to evaluate it, focus on the goal of metrics. The easiest data to collect is not always the most useful. Productivity can be difficult to quantify, especially in high tech environments where one-of-a-kind systems are developed with advanced tools, such as auto-coders. Try not to overlook, for example, the advantage that would be obtained by improving estimating techniques or reducing error rates in delivered systems.

Metrics processes for a Rapid Development environment are not yet well tested. Some suggestions for an initial approach can be found in section 8.

8.0 A Metrics Program for the Rapid Development Process

The application of a metrics process to the Rapid Development model is new and uncharted territory. Based on a combination of experience with Rapid Development projects and metrics programs used in more traditional project models, the project team has identified a proposed initial set of metrics to use in a Rapid Development environment. The methods and rationale for choosing this set are described in the remainder of this section. The Rapid Development Laboratory of the JSC/Aerosciences and Flight Mechanic's Division plans to test this metrics program on a selected project and report on the results.

8.1 Objectives

The objectives for the metrics program during the current fiscal year (FY96) have been:

- Survey NASA/MOD source material on metrics applications for development, maintenance, and project management functions
- Develop recommendations for tailoring metrics for Rapid Development Laboratory (RDL) applications in the near term (i.e. initiate data collection process and limited analysis)
- Recommend COTS software tools to aid the RDL metrics collection and analysis processes
- Support the start-up of the definition, implementation, and analysis processes for metrics data

It was decided early in the project to adopt a currently successful metrics program from the JSC environment, modifying selected metrics and implementing necessary changes for a Rapid Development paradigm.

8.2 Background

A set of development software metrics was adopted by the JSC/Mission Operations Directorate (MOD) during the period from May 1990 to March 1992. The metrics selected were the result of recommendations from a joint MOD and contractor team called CASE (Coordination and Systems Engineering). The resulting metrics were applied on projects across the MOD divisions to orient management discussions, to define correctional actions, and to define and accumulate metrics for new project planning.

MOD expanded their development project metrics task in February 1992 to cover hardware aspects of hardware/software development tasks. The CASE team made additional recommendations to include project evaluation and review techniques for project management applications. The expanded metrics set featured the analysis of project schedule, cost, delivered content, and quality.

The MOD metrics program has been and is being used successfully as a management tool on The Mission Control Center upgrade task at JSC and the development of the Integrated Planning System for Space Station planning and analysis. Both these projects are large efforts requiring incremental development to manage risk and respond to fluctuating budget uncertainties.

The metrics used on the MOD metrics program were adopted as candidate metrics for the RDL RTOP task. They are presented in detail in the following section.

The source documents analyzed from the MOD metrics program include:

*DA3 Software Development Metrics Handbook, JSC-25519, Version 2.1, April 1992

DA3 Software Sustaining Engineering Metrics Handbook, JSC-26010, Version 2.0 December 1992

DA3 Development Project Metrics Handbook, JSC-36112, Version 5.0, March 1993

†DM Process Integrity Metrics Plan, JSC/DM-93-37, July 1995

* DA3... MOD Assistant Director for Program Support
Space Station Ground Division
Shuttle Ground Division
Reconfiguration Management Division

† DM... Flight Design and Dynamics Division

8.3 Candidate Metrics

The metrics associated with the MOD hardware/software development processes are summarized in Tables 1 through 5 on the following pages. They include metrics for development engineering, sustaining engineering, and project management. The metrics have been designed for support in a conventional development environment. The next few sections will present rationale for selection of a subset of these metrics as the early stage of a time phased growth program for Rapid Development metrics. In addition, modifications to the metrics subset will be recommended to adjust for specific variances in the development process engendered by the unique requirements of a Rapid Development process.

It may be noted in Tables 1 through 3 that some metrics are applicable to both development and sustaining engineering functions. This is represented by an ‘X’ in both the ‘Dev’ and Sus’ columns of the appropriate tables. A short description of the metrics may be found under the “description” column. The data which must be collected for a particular metric may be found in the last column, ‘Data Collected During the Reporting Interval’. A short list of metrics acronyms may be found in appendix D.2 on page 84. It is hoped that they will provide a little help navigating the metrics tables.

Table 1. CANDIDATE METRICS - DEVELOPMENT ENGINEERING

CANDIDATE METRICS - DEVELOPMENT ENGINEERING

Dev	Sus	Metric Classification	Description	Data Collected During the Reporting Interval
X	X	Software Size	The SLOC in the system that must be tested and maintained	Total SLOC New SLOC Modified SLOC Reused SLOC Baseline SLOC Deleted SLOC Test baseline SLOC Unmodified SLOC Unmodified Baseline SLOC Ratio of Comments to Total SLOC
X	X	Software Staffing	Number of engineering and first line management personnel involved in system development	Planned staff hours Actual staff hours
X		Software Requirements Stability	Total number of requirements to be implemented for the project	Total A,B,C-level reqts (“shalls”) Cumulative changes
X		Development Progress	Number of modules successfully completed from design through test	Planned and actual “units” designed Planned and actual “units” coded Planned and actual “units” tested Planned and actual “units” integrated
X	X	Computer Resource Utilization	Percent of CPU, disk, memory, and I/O channel utilization	CPU utilization/capacity disk utilization/capacity memory utilization /capacity I/O channel utilization /capacity

Table 2. CANDIDATE METRICS - DEVELOPMENT ENGINEERING(Continued)

CANDIDATE METRICS - DEVELOPMENT ENGINERING (Continued)

Dev	Sus	Metric Classification	Description	Data Collected During the Reporting Interval
X		Test Case Completion	Percent of test cases successfully completed	Planned system integration tests Actual system integration tests
X	X	Discrepancy Report Open Duration	Time lag from problem report initiation to problem report closure	Critical DRs closed in <10 days Critical DRs closed in <30 days Critical DRs closed in <60 days ..... etc.
X	X	Fault Density	Open and total defect density over time	New SLOC Modified SLOC Total DRs written Total DRs closed Active test hours
X		Test Focus	Percent of problem reports closed with a software fix	Total DRs closed Total DRs closed with a single fix
X	X	Software Reliability	Probability that the software works for a specified time under specified conditions	Cumulative critical DRs written Cumulative major DRs written Cumulative minor DRs written Active test hours
X	X	Design Complexity	Number of modules with a complexity greater than an established threshold	Number of modules with McCabe > 10 Number of modules with McCabe > 40 ..... etc.
X	X	Ada Instantiations	Size and number of generic subprograms developed and the frequency of their use within the project	Number of generic units developed Number of instances of the generic unit Generic unit SLOC count

Table 3. CANDIDATE METRICS - SUSTAINING ENGINEERING

CANDIDATE METRICS - SUSTAINING ENGINEERING

Dev	Sus	Metric Classification	Description	Data Collected During the Reporting Interval
	X	Break/Fix Ratio	Ratio of DRs resulting from a DR fix or SR change to the total number DRs+SRs	No. of DRs changed with software fix No. SRs closed with software change DRs closed with a software fix generated with a previous DR fix or SR enhancement
	X	Software Volatility	Number of times a module is changed due to a Service Request	Number of modules changed per release Total modules in release Release date
	X	SR Scheduling	The length of time to close an SR and the effort spent on SR closure	Date of submission Date of availability for release inclusion Date of SR release to facility
	X	Problem/Enhancement Closure	Actual DR and SR closure rate by (sub)system	DRs submitted/closed by (sub)system SRs submitted/closed by (sub)system
	X	Fault Type Distribution	Percent of defects closed by type of fault (e.g. logic, error handling, standards, interface, etc.)	Number of DRs closed by category Number of DRs closed by code Number of DRs closed by fault type Number of DRs closed by process identity
	X	Staff Utilization	Staff effort for DRs by (sub)system Staff effort for SRs by (sub)system	Effort per DR open/closed by (sub)system Effort per SR open/closed by (sub)system

Table 4. CANDIDATE METRICS - PROJECT MANAGEMENT

CANDIDATE METRICS - PROJECT MANAGEMENT *

Metric Classification	Description	Data Collected/Analyzed During the Reporting Interval
Schedule Performance	Milestone Volatility for the Next Year	MVYI = MAM/TM
	Project Critical Path Performance Index	$PCPI_i = ACPMI_i/SCPM_i$
	Reporting Period Milestone Performance Index	$MPI_i = AM_i/SM_i$
	Cumulative Milestone Performance Index	CMPI = CAM/CSM
	Project Schedule Deviation on Critical Path	PAT = (EBT + 4*MLT + EWT)/6
	Earned Value Rate to Completion Rate Index	EVCRI = AER/MER
	Schedule Performance Index	$SPI_i = BCWP_i/BCWS_i$
	Acquisition Performance Index	$API_i = AAM_i/SAM_i$
Cost Performance	Budget Performance Index	$BPI_i = BU_i/AU_i$
	Cumulative Budget Performance Index	CBPI = CBU/CAU
	Staffing Index	$SI_i = 100*(ASRU_i - PSRU_i)/PSRU_i)$
	Cost Performance Index Using EV	CPI = BCWP/ACWP
Cost-Schedule Performance	Project Performance Index	PPI = (MPI + BPI)/2
	Cumulative Project Performance Index	CPPI = (CMPI + CBPI)/2
	Cost Schedule Index Using EV	CSI = (SPI + CPI)/2

* A table of metrics acronyms found in the project management metrics tables may be found in Appendix D.2

Table 5. CANDIDATE METRICS - PROJECT MANAGEMENT (Continued)

CANDIDATE METRICS - PROJECT MANAGEMENT (Continued)

Metric Classification	Description	Data Collected During the Reporting Interval
Delivered Content	Cumulative Risk Performance Index	CRPI = RRK/TRK
	Reviews and DRLIs Performance Index	$RDPI_i = (AMR_i/PMR_i + ADRL_i/PDRL_i)/2$
	Hardware and Software Delivery Performance Index	$HWSWI = K_1*AHW/PHW + K_2*ASLOC/PSLOC$
	Requirements Volatility from Baseline Index	$RVBI = (Changed + Added + Deleted)_{REQTS}$ /Total Baselined Requirements
	Training Performance Index	$TPI_i = ATH_i/PTH_i$
Quality Performance	DRL Rework (RIDs or CRs) Average	$DRLRA_i = (S(RID_{ij})_{j = 1,Ni})/N_i$
	Hardware Development DRs Rate	HWDRR = HWDR/HWUT
	Software Development DRs Rate	SWDRR = SWDR/SWIT
	DR Rate	DRR = DR/TR
	Quality Point Reviews Index	QPRI = AQPR/PQPR
	Average Quality Point Review Score	$AQPRSi = (S(QPS)_{j = 1,Mi})/M_i$
External Influences	Reqts Volatility from Baseline Impact	Estimates of dollar or time impacts due to: changing requirements untimely decisions project dependency items unscheduled work
	Timeliness of Decision and Proj Dep Items	
	Unscheduled Work Impact	
	Subsystems Impacted by Reqts Deviation	
	Requirements Complexity	

8.4 Selected Metrics and Rationale

The development engineering metrics presented in the previous subsection have been selected for early implementation on the Rapid Development project. They are the twelve metrics listed first in the tables. The rationale for their selection may be summarized as follows:

- Adopt useful software/hardware development metrics early in the process.
  - (useful to project development and comparable to existing metric databases)
- Minimize early data collection requirements.
  - (defer sustaining engineering and project management metrics collection/ analysis)
- Provide easy metric project start-up capability.
  - (minimize up-front cost/impact for metrics program initiation)
- Lay the foundation for metrics program growth with minimal rework.
  - (minimal breakage as program grows)

The development engineering metrics from the candidate set are felt to satisfy each of these criteria. These metrics, derived from a successful program in use at NASA/JSC, will be subjected to further analysis in section 8.5 on page 44, where recommendations for their modification will be presented to address problems specific to their application in a non conventional (that is, Rapid Development) process.

The selected metrics are re-presented in Tables 6 and 7. Here the emphasis is refocused on the data collection requirements. The data types and data sources are identified in the second and third columns of the tables and the collected data definitions are expanded slightly in the last column.

The terms ‘Manual Data Form’ and ‘DB (Database) Acquisition’ are common occurrences in the tables. The ‘Manual Data Form’ term refers to the manual collection of data in the early phases of the metrics program, to be replaced in the later stages by an automated electronic procedure. The actual data forms to support the early program data collection requirements are presented in Appendix D. We need to defer their presentation until the recommended modifications to the metrics are discussed since they will impact the actual data collected. The ‘DB Acquisition’ term refers to the capture of data from a paper database in early program phases, transitioning to an automated electronic system linking, say, time card charges in a MIS database to metrics support software for project management metrics like earned value. The time card charges would reflect proper allocation to a Work Breakdown Structure and, therefore, cost data distributions.

Table 6. DATA COLLECTION REQUIREMENTS

DATA COLLECTION REQUIREMENTS

Metric	Data Type	Data Source	Collected Data
Software Size	Manual Data Form	Unit Lead(s)	Total SLOC New SLOC Reused SLOC Baseline SLOC Deleted SLOC Unmodified SLOC Test Baseline SLOC Unmodified Baseline SLOC Ratio of Comments to Total SLOC
Software Staffing	Manual Data Form DB Acquisition	Project Lead Project Database	Actual Staff Hours for month Planned Staff Hours for month
Software Require-ments Stabil-ity	DB Acquisition Manual Data Form	Project Database Project Lead	Total Level A, B, C Requirements Cumulative Changes to Requirements
Development Progress	DB Acquisition Manual Data Form	Project Database for planned Project Lead for Actual	Units Designed (planned and actual) Units Coded (planned and actual) Units Tested (planned and actual) Units SSATed (planned and actual)
Computer Resource Utilization	DB Acquisition Manual Data from Project System Personnel	Project Database for Capacity Sys Support for Utilization	Resource Unit Capacity Resource Unit Utilization Resources = CPU, Disk, Memory, I /O Channel
Test Case Completion	DB Acquisition Manual Data	Project Database Project Lead	Planned System Integration Tests Actual System Integration Tests

DATA COLLECTION REQUIREMENTS (Continued)

Metric	Data Type	Data Source	Collected Data
Discrepancy Report Open Duration	DB Acquisition	DR/SR Tracking System	Critical DRs closed in <10 days Critical DRs closed in <30 days Critical DRs closed in <60 days ..... etc.
Fault Density	Manual Data Form DB Acquisition Manual Data Form	Unit Lead(s) DR/SR Tracking System Project Lead	New SLOC Modified SLOC Total DRs written Total DRs closed Active test hours
Test Focus	DB Acquisition	DR/SR Tracking System	Total DRs closed Total DRs closed with a single fix
Software Reliability	DB Acquisition Manual Data Form	DR/SR Tracking System Project Lead	Cumulative critical DRs written Cumulative major DRs written Cumulative minor DRs written Active test hours
Design Complexity	Manual Data Form	Unit Lead(s)	Number of modules with McCabe > 10 Number of modules with McCabe > 40 ..... etc.
Ada Instantiations	Manual Data Form	Project/Unit Lead(s)	Number of generic units developed Number of instances of the generic units Generic unit SLOC count

Table 7. DATA COLLECTION REQUIREMENTS (Continued)

8.5 Modification of the Selected Metrics

The metrics selected thus far are targeted at the classical software development methods. They need to be modified for adaptation to a Rapid Development environment. The major areas of impact so far identified are:

- Risk management and amelioration underlie much of the Rapid Development approach. How do we specifically address metrics comparisons which may be significantly impacted by a Rapid Development methods?
- The structure of Rapid Development methods introduces multiple cycles in the design to test phases. The classic metrics should be amended to reflect the impact of a spiral development model.
- Project management metrics for conventional development applications are centered around concepts designed for conventional development. How do we define a Rapid Development analogue which may be predicated on a different fundamental metric set (e.g. Pruns vs. SLOC, etc.)?
- The metrics need to be extended to include hardware-in-the-loop aspects of the life cycle development.

A brief discussion of these problems and recommendations for modification of the selected metrics to accommodate these problems in a Rapid Development environment are presented in the following subsections.

8.5.1 Requirements Uncertainties

A major reason for using an Evolutionary development model in a Rapid Development paradigm is the risk management offered by the “build a little, test a little, fly a little” process. The project is started without full knowledge of user requirements (on both the users and developers part) and as the development process evolves, the developers, in concert with the users generate an increasingly mature and more complete user requirements understanding and definition, which, in turn feeds the next development cycle. Thus, there is a continuing evolution and growth of requirements during the project. These changes are not to be confused with the normal parlance of a “Change Request” which represents a requirements change from a baseline defined at or near project start. The “discovered requirements” resulting from an increasingly better understanding of the user needs are tracked as Requirements Changes (RC). Hence, the following modifications to the development metrics are recommended to isolate the impact of uncertainty in the user requirements on the project’s system development process

- Add RC tracking to the DR Open Duration metric
- Add RC tracking to the Fault Density metric
- Add RC tracking to the Test Focus metric
- Add RC tracking to the Software Reliability metric
- Add RC tracking to the Software Requirements Stability metric

8.5.2 Function Point Measurements

Function Points are a logical (functional) unit measure of software functions of the system as seen by the user. It counts internal logical files, external interface files, external inputs, external outputs, and external inquiries. The results are then adjusted based on the complexity of the system defined by a set of general characteristics. Its power comes from the emphasis on the external point of view. Because effort and cost estimation based on Function Point Analysis does not depend on language, operating system, platform, or development process, it avoids many of the problems that arise from the use of source-lines-of-code (SLOC). Additionally, the Function Point counts are available in the early stages of development. The following modification to development metrics are recommended to gain the benefits of Function Point Analysis

- Add Function Point measurement to the Software Size metric

8.5.3 Hardware-in-the-Loop

RDL projects usually feature software development functions complemented by hardware-in-the-loop functions. The hardware related functions are usually centered on:

- Hardware purchase - direct cost of purchasing the hardware elements
- Hardware acquisition - process costs, not actual hardware costs
- Familiarization - personnel learning new hardware and adapting it to the RDL
  - Training
  - Installation
  - Checkout
  - Acceptance
- Integration - making the hardware work in the software/hardware construct
  - Hardware development (emulators, prototypes, device drivers, etc.

The cost associated with these items should be tracked separately from software development efforts by establishing a WBS (work breakdown structure) which uniquely recognizes hardware related costs. Additionally, the following modifications to the development metrics are recommended to isolate the impact of hardware-in-the-loop:

- Add separately tracked hardware related cost to the Software Staffing metric
- Add separately tracked hardware related cost to the Development Progress metric
- Add separately tracked hardware related cost to the Computer Resource Utilization metric
- Add separately tracked hardware related cost to the Design Complexity metric

8.5.4 Metrics for Autocode Generated Software Testing

This is a problem identified late in the FY and has not yet received the level of attention of the previously discussed problems. It will be analyzed in more detail early in FY 97. Here, we present some of the questions which will drive the analysis.

- What problems have been experienced in the RDL with use of Autocode?
- Can you quantify the “quality testing” concept for Autocode (DRs per SLOC, RCs per

Function Point, Fault Density DRs and RCs, Design Complexity of Autocode generated code, et al)?
- Matrix_x allows creation of pseudo-code directly inside Blockscript constructs. This is an excellent opportunity for equivalent “desk checking” of Autocode output at the Ada or C package levels. Can this be formalized? Does it have a payoff?
- Autocode quality problems are mentioned in the McDonnell Douglas Matrix_x Training class documentation (Doane and Weed, March 1995) where code complexity is cited as a major contributing factor. Can we identify potentially complex packages and either manually code them or intensify code review procedures for those cases? If so, this would imply an integration process with Autocode and manually coded segments. The Space Station Multi-Rigid Body Simulation (SSMRBS) project is currently experiencing problems with just such an integration process.
- Matrix_x tends to support top-down development methodology as do Rapid Development paradigms. Would a combination of top-down for system level and bottom-up for subsystem or package level result in needed emphasis on smaller and less complex Autocoded blocks and result in a higher overall quality code?

8.6 Data Definition

Definitions for the modified selected metrics are exhibited in appendix D.1 on page 69. The appendix also adds any supporting definitions required to understand the metrics, units of measure, identification of reporting level requirements, a summary of the data collected for each metric, and some short explanatory notes where appropriate. The information in the metrics descriptions are based on the data contained in the source documents of section 8.2 on page 34. They have been modified and expanded, however, to reflect adaptation to a Rapid Development environment.

8.7 COTS Software for Metrics Support

The Checkpoint metrics software system, marketed by Software Productivity Research has been selected to provide metrics collection, analysis, and presentation support. The major capabilities and features of Checkpoint include:
- Provide measuring capabilities for a full range of software project elements at a user-definable level of detail.
- Assess a full spectrum of software attributes against industry standards for cost, quality, schedules, and productivity.
- Provides aggregate data across selected projects for the establishment of benchmarks for quality and productivity.
- Performs ‘what if’ analyses of languages, skills, methodologies, CASE tools and other variables.
- Provides an estimating package based on over 6000 scientific, industrial, and commercial projects.
- Provides import/export capability with COTS spreadsheets, database systems, and MIS applications.
- Permits flexible Work Breakdown Structures for precise project task and deliverables

description.
- Function point and source-lines-of-code software size metrics are both available.
- Customizable data collection templates and records.
- Functions in a Unix environment supported by X-Windows on a Sun SPARC workstation

8.8 Timeline for Metrics Implementation

Implementation of a full metrics program tailored to the requirements of the RDL will take approximately 2-4 years depending on funding and exigency of RDL needs. The primary focus of the RDL near term metrics planning is the Rapid Development RTOP. Therefore, the metrics implementation plan will be driven by the near term needs and will reflect an incremental implementation plan designed to support the RTOP metrics requirements but with an improvement and expansion path capable of easily supporting growth in the metrics related activities to suit a more mature metrics program of the future. The major stages of the long term metrics plan are:

- Identify a baseline software development metrics set.
- Modify the software development metrics set to reflect the hardware-in-the-loop characteristics of the RDL.
- Modify the software development metrics set to accommodate a Rapid Development “evolutionary” paradigm (requirements uncertainties, function point sizing, and code quality of Autocode generated software.
- Execute the metrics capability established in the first three stages to support the Rapid Development RTOP.
- Expand the metrics program to introduce sustaining and project management metrics.
- Adjust the metrics program to reflect lessons learned in the RDL environment.
- Automate the data collection, analysis, statusing, and management support functions.

The first three stages of the metrics implementation plan are targeted to be complete by the end of September 1996. The fourth stage, the use of the metrics program to support a Rapid Development test project for the RTOP, is targeted for 1997. The remaining stages will be executed as funding and needs dictate. A fully mature metrics program, featuring automated data collection, data analysis, and project management support on both project specific and cross-project data, will require two to four years for full realization and is outside the scope of the RTOP.

Appendix A: References

A.1 Text Books

W.W. Agresti, *New Paradigms for Software Development*, IEEE Computer Society Press, 1986.

S.J. Andriole, *Rapid Application Prototyping: The Storyboard Approach to User Requirements Analysis*, QED Technical Publishing Group, 1991.

W. Bischofberger & G. Pomberger, *Prototyping Oriented Software Development: Concepts and Tools*, Springer-Verlag, 1992.

B.H. Boar, *Application Prototyping: A Requirements Strategy for the 80's*, John Wiley & Sons, 1984.

J.L. Connell & L. Shafer, *Structured Rapid Prototyping: An Evolutionary Approach to Software Development*, Yourdon Press Computing Series, 1989.

C. Gane, *Rapid System Development: Using Structured Techniques and Relational Technology*, Prentice Hall, 1989.

S. Hekmatpour & D. Ince, *Software Prototyping, Formal Methods and VDM*, Addison-Wesley, 1988.

K.E. Lantz, *The Prototyping Methodology*, Prentice-Hall, 1985.

M. Mullin, *Rapid Prototyping for Object-Oriented Systems*, Addison-Wesley, 1990.

M.F. Smith, *Software Prototyping: Adoption, Practice and Management*, McGraw-Hill Book Co. (UK), 1991.

A.2 World Wide Web Sites

http://www.lehigh.edu/kaz2/public/www-data/rp/rp.html RAPID PROTOTYPING

http://www.questicn.com/questicn/radnotes.htm Rapid Prototyping and Evolution

http://www.cranfield.ac.uk/aero/rapid/rapid_prot.html Rapid Prototyping

http://cadserv.cadlab.vt.edu/bohn/RP.html The Rapid Prototyping Resource Center

http://www.iao.fhg.de/Library/rp/OVERVIEW-en.html Virtual Library on Rapid Product Development

http://www.lookup.com/homepages/56694/scsq/scsq1.htm Southern California Software Quality

http://stsc.hill.af.mil/www/ Software Technology Support Center Home Page

A.3 Articles & Papers

J.A. Airst & E.J. Norgren, "Rapid Prototyping Using the VMEbus in an Open Systems Architecture Real-Time Environment", *Proceedings The Fourth International Workshop on*

Rapid System Prototyping: Shortening the Path from Specification to Prototype, June 28-30, 1993, Research Triangle Park, NC, pp. 89-99.

S. Alexander, J. Koehler, J. Stolzy, & M. Andre, "A Mission Management System Architecture for Cooperating Air Vehicles"; *Proceedings of the IEEE 1994 National Aerospace and Electronics Conference (NAECON)*, May 23-27, 1994, Dayton, OH; pp. 156-163.

J.M. Ball & E.J. Riel, "Strategic Avionics Technology Program: Rapid Prototyping & Integrated Design Application Studies, Final Report"; NAS9-18877, February 1995.

J.M. Ball, D.C. Weed & E.J. Riel, "Strategic Avionics Technology Program: Rapid Prototyping & Integrated Design Application Studies, White Paper, Rapid Development Process"; NAS9-18877, February 1995.

H.J. Bender, "Alternative Life Cycle Models for Spacecraft Mission Operations Software Development"; *Proceedings of the ESA Symposium: Ground Data Systems for Spacecraft Control*, June 26-29, 1990, Darmstadt, Germany; pp. 443-448.

E.H. Bersoff & A.M. Davis, "Impacts of Life Cycle Models on Software Configuration Management"; *Communications of the ACM*, August 1991, vol. 34 no. 8, pp. 104-118.

B.W. Boehm, "A Spiral Model of Software Development and Enhancement", *Tutorial: Software Engineering Project Management (R.H. Thayer, ed.)*, Computer Society Press of the IEEE, 1988, pp. 128-142.

B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

P. Branton, "More Haste, Less Speed"; *Computer Weekly*, September 7, 1995, p. 37.

A.S. Brown, "Rapid Prototyping-Parts Without Tools"; *Aerospace America*, vol. 29, August 1991, pp. 18-23.

C. Burns, "REE-A Requirements Engineering Environment for Analyzing and Validating Software and System Requirements"; *Proceedings of the Fourth International Workshop on Rapid System Prototyping: Shortening the Path from Specification to Prototype*, June 28-30, 1993, Research Triangle Park, NC; pp. 188-193.

C. Comaford, "Don't Let Haste Plague Your RAD"; *PC Week*, January 15, 1996, vol. 13 no. 2, p. 18.

J. Connell & G. Wenneson, *Software Engineering Guidebook*, NASA-CR-177625.

L. Constantine, "Under Pressure"; *Software Development*, October 1995, vol. 3 no. 10, p. 112.

L. Constantine, "Re: Architecture"; *Software Development*, January 1996, vol. 4 no. 1, p. 88.

A.M. Davis, "Operational Prototyping: A New Development Approach"; *IEEE Software*, vol. 9 no. 5, September 1992; pp. 70-78.

E.L. Duke, R.W. Brumbaugh & J.D. Disbrow, "A Rapid Prototyping Facility for Flight Research in Advanced Systems Concepts", *Computer*, vol. 22 no. 5, May 1989, pp. 61-66.

A. Dunne, "Rapid Application Development"; *Software Development*, March 1995, vol. 3 no. 3, p. 55.

J.J. Ensell, “Rapid Prototyping for ASIC Designs”, *Electronic Engineering Times*, June 19, 1995, no. 853, p. 82.

G.E. Fisher, “Rapid System Prototyping in an Open System Environment”; *Proceedings of the Fifth International Workshop on Rapid System Prototyping: Shortening the Path from Specification to Prototype*, June 21-23, 1994, Grenoble, France, pp. 213-219.

B.R. Givens, “Object-Oriented Applications in a Rapid Prototyping Environment”; *Proceedings of the IEEE 1994 National Aerospace and Electronics Conference (NAECON)*, May 23-27, 1994, Dayton, OH; pp. 814-819 vol. 2.

B.R. Givens & T.R. Hoerig, “A Scalable Architecture for the Rapid Prototyping of Aircraft Cockpits”, *Proceedings of the IEEE 1993 National Aerospace and Electronics Conference, NAECON 1993*, May 24-28, 1993, Dayton, OH; vol. 1 pp. 523-528.

I. Glickstein & P. Stiles, “Application of AI Technology to Time-Critical Functions”; AIAA/IEEE Digital Avionics Systems Conference, October 17-20, 1988, San Jose, CA

E.M. Halbfinger & B.D. Smith, “The Range Scheduling Aid”; *Proceedings of the NASA JSC Fourth Annual Workshop on Space Operations Applications and Research (SOAR 90)*, Houston, TX; pp. 280-284.

M. Hanna, “Farewell to Waterfalls”; *Software Magazine*, May 1995, vol. 15, no. 5, p. 38.

J. Harper, “RAD Roundup: A Down and Dirty Tour of the Fast-Paced World of Rapid Application Development”; *HP Professional*, May 1995, vol. 9 no. 5, p. 30.

H.-J. Herpel, M. Held, & M. Glesner; “MCEMS Toolbox: A Hardware-in-the-Loop Simulations Environment for Mechatronic Systems”; *MASCOTS ‘94, Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, January 31- February 2, 1994, Durham, NC; pp. 356-357.

H.-J. Herpel, N. Wehn, & M. Glesner, “RAMSES-A Rapid Prototyping Environment for Embedded Control Applications”, *Proceedings of the Second International Workshop on Rapid System Prototyping: Shortening the Path from Specification to Prototype*, June 11-13, 1991, Research Triangle Park, NC; pp. 27-33.

K. Jones & T. Shepard, “Focusing Software Requirements Through Rapid Prototyping”; *Proceedings of the 1994 Canadian Conference on Electrical and Computer Engineering,* September 25-28, 1994, Halifax, NS, Canada; pp. 629-632 vol. 2.

R.E. Jones, “The Advanced Avionics System Development Laboratory”; *Proceedings of the IEEE/AIAA 10th Digital Avionics Systems Conference*, October 14-17, 1991, Los Angeles, CA; pp. 242-247.

A.F.U. Khan, “Mapping the Expert Systems Development Process to the Department of Defense Life-Cycle Model”; *Proceedings, IEEE Conference on Managing Expert System Programs and Projects*, September 10-12, 1990, Bethesda, MD; pp. 31-40.

J. King, “Consult Users Early, Often: Price Waterhouse System Goes Up in Just Four Months”; *Computerworld*, October 16, 1995, vol. 29 no. 42, p. 77.

E.M. Lancaster & D.A. Petri, "Program Life Cycle and the System Engineering Process"; JSC-49037, November 1993.

R.-J. Lea, S. Chen, & C.-G. Chung, "On Generating Test Data From Prototypes"; *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, September 11-13, 1991, Tokyo, Japan; pp. 345-350.

B.M. Leiner & J.R. Weiss, "Telescience Testbedding: An Implementation Approach"; Research Institute for Advanced Computer Science, NASA Ames Research Center, NASA-CR-185429, 1988.

D.S. Linthicum, "The End of Programming"; *Byte*, August 1995, vol. 20 no. 8, p. 69.

D.S. Linthicum, "RAD: Fast Food for Application Hunger"; *Open Computing*, December 1995, vol. 12 no. 12, p. 48.

M. Lubars, C. Potts, & C. Richter, "A Review of the State of the Practice in Requirements Modeling", *Proceedings of IEEE International Symposium on Requirements Engineering,* Jan. 4-6, 1993, San Diego, CA; pp. 2-14.

M.D. Lubars, "Reusing Designs for Rapid Application Development"; *ICC 91 International Conference on Communications Conference Record*, June 23-26, 1991, Denver, CO; vol. 3 pp. 1515-1519.

L. Luqi & R. Steigerwald, "Rapid Software Prototyping"; *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, January 7-10, 1992, Hauai, HI; vol. 2 pp. 470-479.

Luqi, "Research Aspects of Rapid Prototyping"; U.S. Naval Postgraduate School, report number NPS52-87-006, 1987.

D. Lyons, "Choosing the Right Tool"; *InfoWorld*, May 1, 1995, vol. 17 no. 18, p. 49.

R.P. Meyer, R.J. Landy & D.J. Halski, "ICAAS Piloted Simulation Evaluation"

M.J. Miedlar & W. Koenig, "Automatic Code Generation for Aerodynamic/Math Models"; *Proceedings of the IEEE 1994 National Aerospace and Electronics Conference (NAECON)*, May 23-27, 1994, Dayton, OH; vol. 2 pp. 839-844.

D. Millington & J. Stapleton, "Developing a RAD Standard", *IEEE Software*, vol. 12 no. 5, September 1995; pp. 54-55.

H. Mirab & F. Tubb, "First Flight Vehicle Controlled by Computer Generated Software"; *Dynamics and Control of Structures in Space: Proceedings of the 2nd International Conference*, September 6-10, 1993, Cranfield, UK; pp. 653-664.

H. Mirb, "Rapid Prototyping for Real-Time Systems Design"; *Proceedings of the IEE Colloquium on High Accuracy Platform Control in Space*, June 14, 1993, London, UK; pp. 13/1-5.

C.R. Moffitt, II & Luqi, "A Language Translator for a Computer Aided Rapid Prototyping System"; U.S. Naval Postgraduate School, report number NPS52-88-021, 1988.

J. Mullin, "Rapid Iterative Systems Engineering (RISE)"; *Proceedings of the IEEE Colloquium on Aspects of Systems Integration*, November 12, 1993, London, UK; pp. 4/1-4.

N.C. Olsen, "Designing a Real-Time Platform for Rapid Development"; *Proceedings of the IEEE Workshop on Real-Time Applications*, July 21-22, 1994, Washington, DC; pp. 70-75.

R. Olson, "System Architecture for Process Integration"; *ICSI '92 Proceedings of the Second International Conference on Systems Integration*, June 15-18, 1992, Morristown, NJ; pp. 240-246.

D.F. Oresky & C.W. Haapala, "Verification and Validation in an Iterative Software Development Environment"; *Proceedings of the Fourth International Workshop on Rapid System Prototyping: Shortening the Path from Specification to Prototype*, June 28-30, 1993, Research Triangle Park, NC; pp. 57-67.

M.B. Ozcan & J. Siddiqi, "A Rapid Prototyping Environment Based on Executable Specifications"; *Proceedings of 1994 IEEE Region 10's Ninth Annual International Conference Theme: Frontiers of Computer Technology*, August 22-26, 1994, Singapore; vol. 2 pp. 790-795.

M.B. Ozcan & J.I.A. Siddiqi, "Validating and Evolving Software Requirements in a Systematic Framework", *Proceedings of the First International Conference on Requirements Engineering*, April 18-22, 1994, Colorado Springs, CO; pp. 202-205.

C. Palmer, D. Burwen, E. Finnie, "RASSP Cuts Development Time and Cost"; *Electronic Engineering Times*, July 17, 1995, no. 857, p. 48.

B.K. Patel, V. Litchfield, D. Tamanaha, & A. Davis, "Real Time Systems/Software Methodologies for Large Aerospace Systems", *1991 IEEE Aerospace Applications Conference Digest*, Februrary 3-8, 1991, Crested Butte, CO; pp. 2/1-9.

M.C. Paulk, C.V. Weber, S.M. Garcia, M. Chrissis, & M. Bush, *Key Practices of the Capability Maturity Model, Version 1.1*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania; February 1993.

D. Pesek, "Rapid Development Lab Configuration Management Plan"; internal document, October 1995.

A. Radding, "Managing RAD Programmers: Conventional Management Techniques Often Don't Work"; *InfoWorld*, October 23, 1995, vol. 17 no. 43, p. 74.

S. Rahmani, A.G. Stone, W.S. Luk, & S.M. Sweet, "Rapid Prototyping via Automatic Software Code Generation from Formal Specifications: A Case Study"; *1992 IEEE Aerospace Applications Conference Digest*, February 2-7, 1992, Snowmass, CO; pp. 95-105.

S. Rahmani, "A Software-First Methodology for Definition and Evaluation of Advanced Avionics Architectures", *Proceedings IEEE/AIAA/NASA 9th Digital Avionics Systems Conference*, October 15-18, 1990, Virginia Beach, VA, pp. 283-288.

Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Progress Report, April-May 1993"; McDonnell Douglas TM-6.23.07-24; June 30, 1993.

Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Lessons Learned Report"; McDonnell Douglas TM-0009-01 enclosure 1; January 28, 1994.

Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Assurance and Test Report"; McDonnell Douglas TM-0009-01 enclosure 2; January 28, 1994

Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Configuration Management Plan"; McDonnell Douglas TM-0009-01 enclosure 3; January 28, 1994

Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Trip Report, Summary of Trip to MDA-Huntington Beach"; McDonnell Douglas TM-0009-01 enclosure 4; January 28, 1994

J.P. Reilly & E. Carmel, "Does RAD Live Up To the Hype?"; *IEEE Software*, vol. 12 no. 5, September 1995; pp. 24-26.

K. Rizman, I. Rozman, & D. Verber, "The CASE Tool RPT Supporting the Rapid Prototyping Approach to Software Development", *6th Mediterranean Electrotechnical Conference Proceedings*, May 22-24, 1991, LJubljana, Slovenia, vol. 2 pp. 1033-1036.

K. Rizman, I. Rozman, & D. Verber, "RPT: A CASE Environment Supporting the Rapid Prototyping Approach to Software Development"; *Proceedings of the Advanced Computer Technology, Reliable Systems and Applications 5th Annual European Computer Conference (CompEuro) '91*, May 13-16, 1991, Bologna, Italy; pp. 208-212.

R.B. Rowen, "Software Project Management Under Incomplete and Ambiguous Specifications"; *IEEE Transactions on Engineering Management*, vol. 37 no. 1, February 1990; pp. 10-21.

W.W. Royce, "Managing the Development of Large Software Systems"; *Proceedings of IEEE WESCON, 1990*, pp. 1-9.

M. Schrage, "Facilitation: Powerful Job with a Wimpy Name"; *Computerworld*, February 12, 1996, vol. 30 no. 7, p. 35.

B.J. Schroer, F.T. Tseng, S.X. Zhang, W.S. Dwan, *Automatic Programming of Simulation Models, Task 3 Final Report*; NASA-CR-184221, 1990.

J.H. Shackelford, J.D. Saugen, M.J. Wurst, & J. Adler, "The Development of an Autonomous Rendezvous and Docking Simulation Using Rapid Integration and Prototyping Technology" (abstract only); *NASA Automated Rendezvous and Capture Review*, 1993.

J.M. Smith, "In This Game, the Hottest Tools are Totally RAD"; *Government Computer News*, March 20, 1995, vol. 14 no. 6, p. 76.

C. Smyrniotis, "Rapid Prototyping: A Cure for Software Crisis"; *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, January 2-5, 1990, Kailua-Kona, HI; pp. 202-210, vol. 2.

W.H. Spuck, "The Rapid Development Method", Jet Propulsion Laboratory, California Institute of Technology; July 1992.

M.B. Srivastava, T.I. Blumenau, & R.W. Brodersen, "Design and Implementation of a Robot Control System Using a Unified Hardware-Software Rapid-Prototyping Framework"; *Proceedings of the IEEE 1992 International Conference on Computer Design: VLSI in Computers and Processors (ICCD '92)*, October 11-14, 1992, Cambridge, MA; pp. 124-127.

A. Stone, C. Scott, & S. Rahmani, "On Definition and Use of Systems Engineering Processes, Methods and Tools", *1993 IEEE Aerospace Applications Conference Digest*, Jan. 31-Feb. 5, 1993, Steamboat, CO, pp. 197-205.

D.Y. Tamanaha, "Structured Process Flows (SPFs): A Process Model for Metrics", *1991 IEEE Aerospace Applications Conference Digest*, February 3-8, 1991, Crested Butte, CO, pp. 3/1-12.

D.Y. Tamanaha & P.J. Bourgeois, "Rapid Prototyping of Large Command, Control, Communications and Intelligence C^3I Systems"; *1990 IEEE Aerospace Applications Conference Digest*, February 4-9, 1990, Vail, CO; pp. 253-263.

J.D. Thornton, "Engineering Work Station for Rapid Proto-typing of Real Time Display Systems"; *Proceedings of the IEEE/AIAA 11th Digital Avionics Systems Conference*, October 5-8, 1992, Seattle, WA; pp. 57-61.

C. Tristram, "People Power"; *PC Week*, January 15, 1996, vol. 13 no. 2, p. 13.

J. Uhde & D. Weed, "Library Reuse in a Rapid Development Environment"; *Proceedings of the AIAA Conference on Computing & Aerospace X*, March 28-30, 1995, San Antonio, TX; pp. 521-530.

J. Uhde-Lacovara, D. Weed, B. McCleary, & R. Wood, "The Rapid Development Process Applied to Soyuz Simulation Production", internal document, 1994

N. Wehn, H.-J. Herpel, T. Hollstein, P. Poechmueller, & M. Glesner, "High-Level Synthesis in a Rapid-Prototype Environment for Mechatronic Systems", *EURO-DAC '92, European Design Automation Conference*, September 7-10, 1992, Hamburg, Germany, pp. 188-193.

R. Weston, "Pushing the Limits of RAD"; *Open Computing*, June 1995, vol. 12 no. 6, p. 30.

Appendix B: Rapid Development Glossary

B.1 Rapid Development Lexicon

The following is a list of terms and their definitions which may be found with some regularity in the literature addressing various topics of a Rapid Development paradigm. There are four sources from which most of the definitions for the terms are derived

- JSC Engineering Directorate; Aerosciences and Flight Mechanics Division; Guidance, Navigation, and Control Rapid Development Laboratory database and experience
- JPL Technical, Commercial and Industrial database and experience
- Technical, Commercial, and Industrial RDM database and experience in the open literature
- IEEE Software Engineering Standards Collection (1994 Edition)

Emphasized text in the definition field of the Glossary indicates that a definition may be found for the emphasized text in the Glossary

TBD signifies the entry is a placeholder to be expanded in a later delivery

AC-100	Hardware in the loop simulation processor (tests **AutoCode** output)
ASDS	Advanced Simulation Development System - Generic trajectory generation and **GN&C/P** simulation tool developed by McDonnell Douglas featuring large reusable libraries of engineering models, utilities, and processes in the Ada language
AutoCode	**MATRIX$_X$** tool for automated translation of **SystemBuild** block diagrams into Ada or C code
Build	Complete integrated and tested, configuration controlled version of system - successive builds.
CASE	Computer Aided Software Engineering - The use of computer based tools to aid in the software engineering process including software design, requirements tracing, code production, testing, document generation, etc.
CI	Configuration Item - An aggregation of hardware and/or software that is designated for **configuration management** (CM) and treated as a single entity in the configuration management process
CM	Configuration Management - A discipline applying technical and administrative direction to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements
COCOMO	Constructive Cost Model - Software cost estimation model based on a large database of commercial, industrial, and military software

	applications derived from the book “Software Engineering Economics” by Barry Boehm
concurrent	The occurrence of two or more activities within the same interval of time achieved by either interleaving the activities or by their simultaneous execution
COTS	Commercial off-the-shelf - purchased software tools usually used in the **SEE**
CSC	Computer Software Component - A decomposition of **CSCI**s. May be composed of other **CSC**s or **CSU**s
CSCI	Computer Software Configuration Item - An aggregation of software components that satisfy some end-user function
CSU	Computer Software Unit - The lowest level of **CSCI** decomposition
Deslevs	**Prun** attributes from as developed detailed design
DID	Data Item Description - Essentially a deliverable document
DocumentIt	**MATRIX$_X$** automated documentation and debugging tool providing support for Framemaker, Interleaf, and standard ASCII environments
DOD	Department of Defense
Domain Experts	In the **Rapid Development Model** domain experts provide technical expertise across the required range of technical disciplines in the project. They are members of the **Integrated Project Team** and perform the detailed development functions
Dropspecs	Drop specifications in **Spiral Development Model**
Evolutionary Dev Model	The evolutionary development model features the same strategic basis as the **incremental development model** but differs from it in acknowledging that the user need is not fully understood and all requirements cannot be defined up front. The user needs and system requirements are thus only partially defined up front and refined in each succeeding **build**
Evolution Cycle	A **Rapid Development life** cycle phase resulting in increasingly mature **builds** and leading to the final product at the last cycle
Exit Conditions	Requirement conditions to be met at each milestone
FCA	Functional Configuration Audit - An audit conducted to verify that a configuration item has been completed satisfactorily
FQR	Formal Qualification Review - The test, inspection, or analytical process by which a group of configuration items comprising a system is verified to have met specific contractual performance requirements

GN&C Guidance, Navigation, and Control

GN&C/P The **GN&C** system combined with the propulsion system

GUI Graphical User Interface - Generic term for utilization of screen data presentation and user input via pointing devices (e.g. mouse) to facilitate user interaction with a software construct

HCE Hardware Connection Editor HCE

HWCI Hardware Configuration Item - An aggregation of hardware components that satisfy some end-user function

Incremental Dev Model A software development technique in which the requirements, design, implementation and testing occur in an overlapping, iterative, manner resulting in incremental completion of the overall product

IDD Interface Design Description - A document defining interfaces between **CSCI**s

ISO 9000 International Organization for Standards method for assessing supplier ability to meet commitments and requirements (International analogue of **SEI CMM**)

Integrated Project Team In the **Rapid Development** paradigm a project dedicated team which assumes ownership of the entire development process and end product. The Integrated Project Team includes all critical domain and systems skills and expertise needed to successfully complete the project.

IV&V Independent Verification and Validation

KPA Key Process Area - **CMM** area of focus

MATRIXx An integrated toolset providing a graphical environment for analysis and development of system requirements, design, development, code, and test over the entire development cycle

Metric Quantitative measure of system size, complexity, cost, quality etc.

OCR Operational Concept Review -.Reviews held to resolve open issues regarding the operations concept for a system

PCA Physical Configuration Audit - An audit conducted to verify that an as built configuration item conforms to the technical documentation that defines it

PMI Project Management Information - the Information required by each **Prun** in addition to the requirements to be used in project management and design

Pruns Projects Units (HW & SW packages, **Superblocks**, models, etc.)

Prototyping A hardware and software development technique in which a

	preliminary version of the hardware/software product is developed to stimulate user feedback, determine feasibility, or investigate timing or other issues in support of the development process
QA	Quality Assurance - TBD
Quality Gates	The set of conditions which must be met to transition from one life cycle phase to the succeeding phase
Rapid Dev Model	A extension of the **Spiral Development Process** where additional tools (such as integrated **SEE** tools like **MATRIXx** are used to speed the process
Rapid Prototyping	A subset of the **Rapid Development** process where an initial prototype version is created, primarily for validating the initial requirements and design concepts
RC	Requirements Change - The reversal of the conventional acronym CR (Change Request) is intended to specify the evolutionary requirements discovered during the Rapid Development evolutionary model exercise as opposed to a conventional new requirement written against the requirements baseline
RDL	Rapid Development Laboratory - NASA/JSC facility for accelerated **GN&C** software development research and applications
Reqlevs	**Prun** attributes from high level requirements (requirements “shalls”)
Reusable Library	Collection of reusable code modules (e.g. utilities, models, etc.)
SDF	Software Development File - A collection of material pertinent to the development of a given software unit or set of related units. Contents typically include the requirements, design, technical reports, code listings, test plans, test results, problem reports, schedules, and notes for the units
SEE	Software Engineering Environment - The hardware, software, firmware, procedures and documentation needed to perform software engineering. Elements may include but are not limited to **CASE** tools, compilers, assemblers, linkers, loaders, operating systems, debuggers, simulators, emulators, documentation tools, and database management systems
SEI	Software Engineering Institute
SLOC	Source lines of code - **metric** for sizing of software products
Spiral Dev Model	Spiral Development Process - An accelerated development process where the system requirements, design, code, test, and integrated test processes are iterated on concurrently rather than being executed sequentially
Statemate	Methods and tools for requirements, development, and validation

STE — Software Test Environment - The facilities, hardware, software, firmware, procedures and documentation needed to perform qualification, and possibly other, testing of software. Elements may include but are not limited to simulators, code analyzers, test case generators, path analyzers, etc. and may include elements used in the **SEE**

Superblocks — TBD

Support Processes — The set of general technical, management, and institutional processes providing support to the development process (e.g. project management, requirements management, configuration management, etc.)

SysemBuild — **MATRIX$_X$** graphical interface tool supporting system design from data flow block diagrams

Test Case Spec. — A document that specifies the test inputs, execution conditions and predicted results for an item to be tested

Test Design — Documentation specifying the details of a test approach for a software feature or combination of features and identifying the associated tests

V&V — Verification and Validation - Determination whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements and conditions imposed by the previous phase, and the final system or component complies with specified requirements

Waterfall Model — Conventional software development model featuring sequential development of the component life cycle phases with requirements established early in the process

Xmath — **MATRIX$_X$** tool for design and analysis of simulations, control systems, and numerical calculations

B.2 MIL-STD-498 Reviews and Documentation

The following is a list of terms associated with documentation and technical/management reviews suggested in MIL-STD-498. The MIL-STD-498 is superseding MIL-STD-2167A and makes many significant changes to accommodate currently evolving Rapid Development processes

The source from which the terms for this Appendix are derived is:

- MIL-STD-498 - Software Development and Documentation, Software Technology Support Center, Hill Air Force Base, Utah,

Emphasized text in the definition field of an entry indicates that a definition may be found for the emphasized text in the elsewhere in the Appendix

TBD signifies the entry is a placeholder to be expanded in a later delivery

498	MIL-STD-498 - **DOD** Standard superseding MIL-STD-**2167A** for the development of systems and software applications. A key feature of the standard is its strategy for accommodating facets of the **Rapid Development model** paradigm
1521B	MIL-STD-1521B - **DOD** Standard for technical reviews and audits invoked by **2167A**
2167A	MIL-STD-2167A - **DOD** Software Development Standard - Establishes uniform requirements for software development applicable throughout the system life. The standard provides the basis for Government insight into contractor software development, testing, and evaluation
CDM	Conventional Development Model - Software development paradigm predicated on a sequential development model of the software/ hardware system. The CDM underlies the structure of the system life cycle in **2167A**
CDR	Critical Design Review - Formal review required by **1521B** to review the detailed designs for each **CSU** and assure the system configuration items meet the specified requirements
CMM	Capability Maturity Model - **SEI** method for assessing supplier ability to meet commitments and requirements (American analogue of **ISO 9000**)
COM	Computer Operation Manual - A **498 DID** containing instructions for operating a computer
CPM	Computer Programming Manual - A **498 DID** containing instructions for programming a computer
CRR	Critical Requirements Review - Reviews held to resolve open issues regarding the handling of critical requirements, such as safety, security, and privacy

DBDD	Database Design Documentation - A **498 DID** describing the design of an associated database
FSM	Firmware Support Manual - A **498 DID** containing instructions for programming firmware devices
IDD	Interface Design Document - A **498 DID** specifying the design of one or more interfaces between one or more software systems and hardware systems
IPR	In-Progress Review - Technical and management reviews scheduled at completion of a **Build** milestone
IRS	Interface Requirements Specification - A **498 DID** containing the requirements for one or more interfaces
OCD	Operational Concept Description - A **498 DID** containing the operational concept for the system
PDR	Preliminary Design Review - Formal review required by **1521B** to review the detailed designs of **CSCI**s and **CSC**s to evaluate the progress, technical adequacy, and risk resolution of the selected design approach for one or more configuration items
SCOM	Software Center Operator Manual - A **498 DID** containing instructions for operators of a batch or interactive software system that is installed in a computer center
SDD	Software Design Description - A **498 DID** containing detailed design for each **CSCI**
SDP	Software Development Plan - A **498 DID** containing the plan for performing the software development activities
SDR	Software Design Review - Reviews held to resolve open issues regarding the architectural design of a **CSCI**, **CSCI**-wide design decisions, and detailed design of a **CSCI** or portion thereof (such as a database)
SIOM	Software Input/Output Manual - A **498 DID** containing instructions for users of a batch or interactive software system installed in a computer center
SIP	Software Installation Plan - A **498 DID** containing the plan for installing the software at the user sites
SPR	Software Plan Review - Reviews held to resolve open issues regarding the **SDP,** STP, SIP, and STrP
SPS	Software Product Specification - A **498 DID** comprised of the executable software, the source files, and information to be used for support
SQPP	Software Quality Program Plan - A description of the plan for

	implementing quality control procedures in a hardware/software system development
SwRR	Software Requirements Review - A review of the requirements specified for one or more software configuration items to evaluate their responsiveness the system requirements
SyRR	System Requirements Review - A review of the completeness and adequacy of the system requirements to evaluate the system engineering process that produced those requirements and to assess the results of system engineering studies
SSDR	System/Subsystem Design Review - Reviews held to resolve open issues regarding the system or subsystems design decisions or the architectural design
SSS	System/Subsystem Specification - A **498 DID** providing documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software system
SRR	Software Requirements Review - Reviews held to resolve open issues regarding the specified requirements for a **CSCI**
SRS	Software Requirements Specification - A **498 DID** presenting the requirements to be met by each **CSCI**
SSDD	System/Subsystem Description - A **498 DID** defining the system and its. partitioning into **HWCI**s and **CSCI**s
SSR	Software Supportability Review - Reviews held to resolve open issues regarding the readiness of the software for transition to the support agency, the software product specifications, the software support manuals, the software version descriptions, and the status of transition preparation and activities
SSRR	System/Subsystem Requirements Review - Reviews held to resolve open issues regarding the specified requirements for a software system or subsystem
STD	Software Test Description - A **498 DID** containing test case descriptions and procedures for qualification testing for one or more software systems
STP	Software Test Plan - A **498 DID** containing the plan for conducting qualification testing
STR	Software Test Report - A **498 DID** containing a record of the formal qualification testing performed on the software system
STrP	Software Transition Plan - A **498 DID** containing the plan for transitioning to the support agency

SUM	Software Users Manual - A **498 DID** containing instructions sufficient to execute a software system
SUR	Software Usability Review - Reviews held to resolve open issues regarding the readiness of the software for installation at user sites, user and operator manuals, software version descriptions, and the status of installation preparation activities
TRdR	Test Readiness Review - Reviews held to resolve open issues regarding the status of the **software test environment,** the status of the software to be tested, and the test cases and procedures to be used for **CSCI** qualification testing or system qualification testing
TRsR	Test Results Review - Reviews held to resolve open issues regarding the results of **CSCI** testing or system qualification testing
SVD	Software Version Description - A **498 DID** which identifies and describes a version of a system or component

Appendix C: Metrics Glossary

The following is a list of terms and their definitions which are associated with the application of metrics to the Software/Hardware development and sustaining engineering processes. There are three sources from which most of the definitions for the terms are derived

- DA3 Software Development Metrics Handbook Version 2.1 - NASA/JSC 25519 April 1992
- DA3 Software Sustaining Engineering Metrics Handbook Version 2.0 - NASA/JSC 26010 December 1992
- DA3 Development Project Metrics Handbook Version 5.0 - NASA/JSC 36112 March 1993

The following Metrics Glossary items do not include project management metrics.

Active Test Time	Elapsed wall-clock time during which software was actively being tested. Active test time does not include time lost due to failure, reconfiguration, or debugging
Actual SLOC	A count of the lines of New and Modified code actually produced during the code production and test phases of a project.
Baseline SLOC	The size of the current operational baseline at project start. Applies to a project to update or modify an existing program.
Break/Fix Ratio	The number of DRs closed with a software fix that were generated as the result of a previous DR fix or CR enhancement divided by the sum of the number of DRs closed with a software fix plus the number CRs closed with a software change
Capacity	Maximum amount of a resource available for use.
Change Request (CR)	A request for system enhancement
Closed Date	The date when the software is returned to operations or is operationally ready. (i.e. the software fix or enhancement is complete and on the floor).
Closure Codes	A classification scheme used to identify how work was completed or submitted on DR requests
Code and Test	Code is the translation of a designed unit into a computer program that can be accepted by a processor. Testing is the exercising (either manually in the case of a walkthrough or electronically through unit test cases or both) of the coded unit
Comment Ratio	The fraction of the number of comment lines in the new and modified software to the SLOC in the new and modified software.
Comment SLOC	A textual string, line, or statement that has no effect on compiler or program operations
Corrective Maintenance	Software changes resulting from Discrepancy Reports

COTS — Commercial-off-the-shelf software that is purchased for a project. COTS software is only included in the software size metric if it is to be maintained by the purchaser

Compilation Unit — The lowest independently compilable software element subject to configuration management.

CSC — Computer Software Component - equivalent to an Ada software package

Computer Resource Utilization — The fraction of time a resource is busy.

Configuration Item — Logically related grouping of units or packages which perform a major function of the system.

Critical DR — A failure that affects the following systems in the manner described:
Development System - Inhibits major processing in more than one area and cannot be circumvented.
Test System - Inhibits one or more applications from being tested, or brings the system to a halt and cannot be circumvented.
Operational Systems - Drastically reduces the usefulness of the system in support of current operations, and cannot be circumvented.
All Systems - Requires reboot of workstation to correct problem.

Data Primitive — A basic data item required to compute a metric value.

Defect — An error in the software.

Deleted SLOC — Existing SLOC that will be removed from the baseline by the completion of the delivery.

Design — The definition of each software unit's control and data structure, interfaces, and lists of accessed data items.

Development Progress — A measure of progress toward design, implementation, and integration of the software.

Discrepancy Report — A notification that a system under test or in operation (i.e. hardware, software, system, operations) has deviated from the behavioral characteristics expected of it. The notification carries a description of the problem, an assessment of the criticality of the problem, and a portion of the system to which the problem is charged.

DR Criticality — The assessed effect the DR has on the continuance of the system activity.(e.g. test, mission support, training. etc.)

DR Density — The total number of DRs written against a piece of software divided by the size of that software.

Error — A human action taken during the design, code, or test of software that results in a fault.

Fault Type — A problem identifier which may be categorized by the closure code of

	the DR, by the DR criticality, or by the taxonomy of faults established for the project.
Failure	The inability of a system or system component to perform a required function within specified limits.
Failure Rate	The cumulative number of failures divided by the cumulative active test time.
Fault Density	The total number of DRs written against a piece of software divided by the size of that software.
Generic Unit	A template that defines a program unit as either a generic subprogram or a generic package.
Integration	The process of combining coded and tested units and configuration items into a system or subsystem.
Instance	Specified subprograms or packages that are obtained by assigning values to the parameters of the generic unit.
Major DR	A failure that affects the following systems in the manner described: Development System - Inhibits major processing or produces erroneous outputs limited to one function Test System - Inhibits an entire processor of an application from being tested or prohibits completion of a test case by blocking other test functions. Operational System - Reduces the usefulness of one or more major system functions used in the current operations, and cannot conveniently be circumvented. All Systems - Logoff/Logon is required to restore operation.
Minor DR	A failure that affects the following systems in the manner described: Development System - Anomalies that slight and can be circumvented Test System - DRs that do not directly affect completion of a test function and are considered to have no effect or to be insignificant in an operations environment. Operational System - DRs that occur during a mission, simulation, or validation period that are considered to have no effect or to be insignificant during that period.
McCabe Complexity	The number of linearly independent paths in a module that, when taken in combination, will generate every possible path.
Modified SLOC	A module or compilation unit is “modified” if it is changed and the change affects less than 20% - 50% of the SLOC.
Module	The lowest level of software compilation subject to configuration management.
New SLOC	Newly developed code or code in a module or compilation unit that has been changed and the changes affect more than 20% - 50% of its

SLOC.

Normalized Active Test Time — Active test time divided by the sum of new plus modified SLOC.

Observed Failure Rate — The cumulative number of failures divided by the active test time.

Open DR Density — The total number of open DRs written against a piece of software divided by the size of that software.

Operational Hour — An hour that the system is directly supporting a primary user.

Progress — A count of the number of DR or CR requests submitted minus the number of DR or CR requests closed during a specified reporting interval.

Project — The set of activities performed to develop a new system or to upgrade an existing system.

Release — The entire software configuration, not just the changed modules.

Resource — An available system component.

Reused SLOC — Those SLOC that are not part of the baseline and exist on a different project but are used on the current project.

Software Reliability — The probability that the software will not cause a failure of a system for a specified time under specified conditions.

Software Requirement — Any "shall" statement in the project's controlling software specification.

Software Staff — All those directly involved in the software development activity, including programmers, testers, and first line managers.

SLOC — An acronym for Source-Line-Of- Code, any non-comment, non-blank carriage return terminated source line of code.

Staffing — The number of hours spent on a project by all those directly involved in the development activity, including programmers, testers, and first line management.

Subsystem — A collection of functionally related software configuration items.

System — A group of interacting, interrelated, or interdependent elements (sub-system or other configuration components) which form a recognized complex whole.

Subprogram — A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program.

Test Baseline SLOC — Those unmodified baseline SLOC that must be retested to verify system operational requirements are met.

Test Case — A specific set of procedures using associated data developed to exercise a particular program path or verify compliance with a specific requirement.

Total SLOC — Total SLOC is defined as (Unmodified SLOC + New SLOC + Modified SLOC - Deleted SLOC + Reused SLOC)

Unit — A collection of modules or compilation units performing a testable function

Unmodified Baseline — The number of current operational baseline SLOC that are not changed during the development effort.

Workload — The collective designation of a system's inputs (i.e. programs, data, commands) over time.

Appendix D: Metrics Definitions, Acronyms and Data Forms

D.1 Definition of the Selected Metrics

This section presents each of the modified selected metrics, adding a short description, any supporting definitions required to understand the metric, units of measure, reporting level requirements, a summary of the data to be collected for each metric, and some short notes where needed.

The information in the metrics descriptions are based on data contained in the source documents of section 8.2 on page 34. They have been modified and expanded, however, to reflect adaptation to a Rapid Development environment.

Metric:

Software Staffing

Description

Tracks planned staffing for the project and progress toward achieving that plan

Supporting Definitions

Software staff - All those directly involved in the software development activity, including programmers, testers, and first line management. The following software development activities are measured by the software staffing metric: software planning, requirements definition, design, coding, test, documentation, configuration management, and quality assurance.

Units of Measure

Software staffing is measured in staff-hours expended per month. All reported hours are counted including overtime (regardless of whether the overtime is compensated or not).

Reporting Levels

The software staffing metric is tracked for each subsystem. The subsystems are accumulated to create a project level report.

Data Collected

Planned and actual staff hours by subsystem per month

Notes

Metric:

Software Size

Description

An estimate of the number of lines of code in a project at its completion. During the code production and test phase of a project, the metric is augmented with a count of lines of code actually produced.

Supporting Definitions

SLOC - A source line of code is counted as any line of program text that is not a comment, regardless of the number of statements or fragments of statements on the line

Module - The smallest independently-compilable software element subject to configuration management. By convention, modules contain only one type of SLOC (e.g. New, Modified, Reused, etc.). This convention is established to simplify correlation of manually developed software size estimates with automated software size estimates.

Baseline SLOC - Baseline SLOC is the size of the operational code in a project. If the operational baseline changes during upgrade development (due to parallel maintenance activities, for example) the Baseline SLOC measure reflects the change. It is used for upgrade projects in which an operational baseline of code is being modified.

Modified SLOC - Modified SLOC is code that exists, but requires changes amounting to less than some percentage change threshold per module, as determined by project management at the beginning of metrics reporting.

New SLOC - New SLOC is newly developed code or code within a module that has been changed by more than an established percent-age change threshold

Reused SLOC - Reused SLOC are those SLOC that exist on a different project and are used without modification on the current project. They are not part of the Baseline SLOC.

Deleted SLOC - Deleted SLOC is existing SLOC that will be remove from the system by the completion of the delivery. Software is counted as Deleted SLOC only if an entire module is deleted. Otherwise the software is counted as New or Modified, based on the scope of the changes to the module.

Total SLOC - Total SLOC is the size of the software that will be sustained at the completion of the project. It is the sum of Baseline SLOC, New SLOC, Modified SLOC, and Reused SLOC.

Test Baseline SLOC - Test baseline SLOC is the amount of code in an upgrade project that is part of the unmodified baseline that must be retested in order to verify that the system operation requirements are maintained.

COTS Software - Commercial off-the-shelf software is software that is purchased for a project. COTS software is only included in the software size metric if it is to be maintained by NASA or if it is to be modified in the course of the project. If the software is to be modified, the unmodified software should be reported as Reused. If the COTS software is to be maintained but is otherwise unmodified, it is incorporated into Baseline with an appropriate annotation

Comment SLOC - A comment SLOC is a textual string, line, or statement that has no effect on compiler or program operations. Omitting or revising comments never changes a program's logic or data structures. In-line comments are included in the comment count. Blank comments are physical lines or statements that have comment designators but contain no other visible text or symbols. They are excluded from counts of source code size.

Actual SLOC - A count of the lines of New and Modified code actually produced during the code production and test phases of a project. As a New or Modified module completes inspection or Unit Test, it is added to the list of modules that comprise the actual SLOC total. Actual SLOC is the sum of the SLOC in these modules.

Units of Measure

The software size metric is measured as a count of SLOC. Each of the SLOC categories are to be reported for each language used (Ada, C, C++,FORTRAN, 4GL, etc.).

Reporting Levels

Software size is reported by release for each subsystem and for the project as a whole

Data Collected

Each of the following code categories are reported by language:

- New SLOC and Function Points
- Modified SLOC and Function Points
- Reused SLOC and Function Points
- Deleted SLOC and Function Points
- Baseline SLOC and Function Points
- Unmodified Baseline SLOC and Function Points
- Test Baseline SLOC and Function Points
- Total SLOC and Function Points
- Ratio of Comments to total SLOC and Function Points

Notes

• Modified, Deleted, Baseline, Unmodified Baseline, and Test Baseline are reported only for projects which feature modifications to an existing baseline. This would include the various cycles of an Evolutionary Development model

• Code SLOC counting should be done with an automated software tool.

Metric:

Software Requirements Stability

<u>Description</u>

The Software Requirements Stability metric is the current number of “shalls” in the project’s controlling documents (e.g. Level “A’s and/or User Detailed Functional Requirements (UDFR)).

<u>Supporting Definitions</u>

Software Requirement - Any “shall” statement in the project’s controlling software specification.

<u>Units of Measure</u>

Software Requirements Stability is measured by the number of “shalls” in the projects controlling document(s).

<u>Reporting Levels</u>

The software requirements stability metric is tracked for each subsystem after the requirement allocation to individual subsystem has been “frozen” at PDR.

<u>Data Collected</u>

Total Level A requirements and cumulative changes to those requirements

<u>Notes</u>

• Requirements counting is best done with a word processing tool in search mode or a requirements management tool like SQL‘

• Cumulative changes to the requirements manifest themselves in Level B and Level C requirements as the project passes from requirements definition through design, code, test, and integration. Many of the “new” requirements generated during these phases will be “discovered requirements” resulting from the increased understanding of user needs revealed during the project

Metric:

Development Progress

Description

The Development Progress metric tracks progress in design, code, test, and integration of the software and hardware elements. Each of the elements are tracked separately.

Supporting Definitions

Code and Test - The translation of a designed unit into a computer program that can be accepted by a processor. Testing is the exercise of the coded unit either manually in the case of an inspection or electronically through unit test cases or both.

Configuration Items - A collection of units that are treated as a single component for the purposes of planning and configuration management.

Design - The definition of each unit's control and data structures, interfaces, and lists of accessed data items.

Integration - The process of combining coded and tested units into a system or subsystem.

Unit - A collection of modules performing a testable function.

Units of Measure

Design progress is reported in terms of unit design completions. Code and Test progress is reported in terms of unit completions. Integration progress is reported in terms of the number of SSAT tests completed

Reporting Levels

Configuration Items - A collection of units that are treated as a single component for the purposes of planning and configuration management. Reporting is at the system and subsystem levels.

Data Collected

• Planned and actual units designed.

• Planned and actual units coded and tested.

• Planned and actual SSAT test cases completed

Notes

Metric:

Computer Resource Utilization

Description

The Computer Resource Utilization (CRU) metric tracks the projected and actual use of the target system's resources.

Supporting Definitions

Capacity - Maximum amount of resource available for use.

Resource - an available system component

Workload - The collective designation of a system's inputs (i.e. programs, data, commands) over time.

Units of Measure

The CPU, memory, disk, and I/O channel utilization are all expressed as a percent of resource capacity.

Reporting Levels

Reporting is at the project level only. Project level reports should address projected/actual CRU for each major type of resource (e.g.host, workstation, compute node, data node, local area network, etc.)

Data Collected

• Projected and actual CPU utilization/capacity.

• Projected and actual memory utilization/capacity.

• Projected and actual disk utilization/capacity.

• Projected and actual I/O channel utilization/capacity.

Notes

Metric:

Test Case Completion

Description

The Test Case Completion metric tracks planned system integration test case completions and progress toward achieving that plan.

Supporting Definitions

Test Case - A specific set of procedures using associated data, developed to exercise a particular program path or verify compliance with a specific requirement.

Units of Measure

The Test Case Completion metric is measured in terms of the number of test case planned and completed.

Reporting Levels

Reporting is at the project level

Data Collected

Planned and actual cumulative system integration test cases completed

Notes

Metric:

Fault Density

Description

The Fault Density metric is a measure of testing adequacy and code quality.

Supporting Definitions

Active Test Time - Elapsed wall-clock time during which software was actively being tested. Active Test Time does not include lost time due to failure, reconfiguration, or debugging.

Discrepancy Report (DR) - A notification that a system under test or in operation has deviated from the behavioral characteristics expected of it. The notification includes a description of the problem, an assessment of he criticality of the problem, and identification of a segment of the system to which the problem is charged.

Requirements Change (RC) - Similar to a Change Request for a Rapid Development methodology but deriving from a “discovered requirement” identified during the design, code, test, and integration phases

Critical DR - A failure that affects the following systems in the stated manner:

- Development System: Inhibits major processing in more than one area and cannot be circumvented.
- Test System: Inhibits one or more applications from being tested, or brings the system to a halt and cannot be circumvented.
- Operational System: Drastically reduces the usefulness of the system in support of current operations, and cannot be circumvented
- All Systems: Requires re-boot of workstation to correct problem.

Major DR - A failure that affects the following systems in the stated manner:

- Development System: Inhibits major processing or produces erroneous output limit to one function.
- Test System: Inhibits an entire processor of an application from being tested or prohibits completion of a test case by blocking other test functions.
- Operational System: Reduces the usefulness of one or more major system functions used in current operations, and cannot be conveniently circumvented.
- All Systems: Logoff/Logon is required to restore the operation.

Minor DR - A failure that affects the following systems in the stated manner:

- Development System: Anomalies that are slight and can be circumvented.
- Test System: DR that do not directly affect completion of a test function and are considered to have no effect or to be insignificant in an operational environment.
- Operational System: DRs that occur during a mission, simulation, or validation period that are considered to have no effect or to be insignificant during that period.

Fault Density - The total number of DRs written against a piece of software divided by the size of that software (in KSLOC)

Normalized Active Test Time - Active Test Time divided by the sum of New plus Modified SLOC. (Test Baseline SLOC is not included in the calculation since it is considered regression testing and inclusion would obscure the desired result)

Units of Measure

Fault Density uses two measures: Total DR density, which is measured in DRs per KSLOC; and Open DR density, which is also measured in DRs per KSLOC

Reporting Levels

Reporting is at the project and subsystem levels.

Data Collected

The Fault Density metric uses the following data primitives:

- New SLOC (as defined in the Software Size metric)
- Modified SLOC (as defined in the Software Size metric)
- Total DRs written (cumulative), all degrees of criticality
- Total DRs closed (cumulative), all degrees of criticality)
- Active Test Hours (ATH)

Auxiliary data generated by the data primitives above include:

- Normalized Active Test Hours = ATH/(New SLOC+Mod SLOC)
- Total DR Density = Total DRs written/(New SLOC+Mod SLOC)
- Open DR Density = (Total DRs written-Total DRs closed) /(New SLOC+Mod SLOC)

Notes

• The same data collected for the code size parameters (SLOC) should be collected for the Function Point parameter

Metric:

Test Focus

Description

The Test Focus metric is the percent of DRs and Requirements Change (RCs) closed with a single software fix. A high Test Focus value is an indicator of effective code test procedures.

Supporting Definitions

Requirements Change - Code enhancement requests not to be confused with classic Change Requests. A fundamental premise of an Evolutionary Development Model in a Rapid Development paradigm is the uncertainty in user needs and their manifestation as “fuzzy” requirements. As development progresses the requirements and user needs become better defined resulting in potential modifications and enhancements to the system specifications. These are not true CRs in the classic sense.

Units of Measure

Test Focus is a percent

Reporting Levels

The Test Focus metric is reported at the project and subsystem levels

Data Collected

Test Focus requires the following data primitives:

- Total DRs and RCs closed (cumulative), all degrees of criticality
- Total DRs and RCs closed with a single software fix, all criticality

Auxiliary data generated by the data primitives above include:

- Test Focus = Total DRs and RCs closed with a software fix * 100/Total DRs and RCs closed

Notes

Metric:

Discrepancy Report and Requirements Change Open Duration

Description

The DR and RC Open Duration metric tracks the amount of time required to close critical DRs or RCs once they are discovered.

Supporting Definitions

None

Units of Measure

The DR Open Duration metric is measured in terms of critical DRs or RCs open for various periods of time

Reporting Levels

The DR or RC Open Duration metrics are reported at the project level and raw data is reported at the subsystem level.

Data Collected

DR Open Duration metric requires the following data primitives:

- Total critical DRs or RCs closed within 0-9 days
- Total critical DRs or RCs closed within 10-29 days
- Total critical DRs or RCs closed within 30-59 days
- Total critical DRs or RCs closed within >60 days

Notes

Metric:

Software Reliability

Description

The Software Reliability metric provides an indication of the expected operational failure rate of the software based on testing failure data.

Supporting Definitions

Active Test Time - Elapsed wall-clock time during which the software was actively being tested. Active Test Time does not include time lost due to failure or reconfiguration, nor does it include debugging time.

Discrepancy Report (DR) - A notification that the system under test or in operation (i.e. hardware software, system, operations) has deviated from the behavioral characteristics expected of it. The notification carries a description of the problem, an assessment of the criticality of the problem, and a portion of the system to which the problem is charged.

DR Criticality - The assessment of the effect the DR has on the continuance of the current system activity (e.g. test, mission support, training. etc.)

Failure - The inability of a system or subsystem component to perform a required function within specified limits.

Observed Failure Rate - The cumulative number of failures divided by the cumulative Active Test Time.

Software Reliability - The probability that the software will not cause a failure of the system for a specified time under specified conditions.

Units of Measure

The Software Reliability metric is measured in units of failures per Active Test Hours for each DR criticality level.

Reporting Levels

Software is reported only at the project level.

Data Collected

The DR Open Duration metric requires the following data primitives:

- Critical DR Failure Rate (Failures per Active Test Hour)
- Major DR Failure Rate (Failures per Active Test Hour)
- Minor DR Failure Rate (Failures per Active Test Hour)

Notes

Metric:

Design Complexity

Description

The Design Complexity metric tracks the software complexity of modules (e.g. subroutines, C++ language functions, function points, etc.) using the Extended McCabe complexity metric.

Supporting Definitions

Extended McCabe Complexity - The number of independent paths in a module that, when taken in combination, will generate every possible path.

Module - The lowest level of software defined and under configuration management

Units of Measure

Design complexity is defined as the number of paths through a module. The metric is reported as a count of modules.

Reporting Levels

Reporting is at the project and subsystem levels.

Data Collected

Number of modules with McCabe metric > 10

Number of modules with McCabe metric > 40

Notes

Metric:

Software Reusability

Description

The Software Reusability metric tracks the reuse of software generic units during project development

Supporting Definitions

Generic Unit - A template that defines a program unit as a generic subprogram or generic package.

Instance - Specific subprograms or packages that are invoked by assigning values to the parameters of a generic unit.

Units of Measure

The Software Reusability metric is measured in:

- the number of generic units developed
- the number of instances of the generics (both “new” and “withed”)
- the total number of generic unit SLOC

Reporting Levels

Generic unit development and utilization is reported only at the project level.

Data Collected

Software Reusability requires the following data primitives

- Number of generic units designed, coded, and tested
- Number of instances of the generic units
- Generic unit SLOC count

Notes

D.2 Metrics Acronyms

AAM	achieved acquisition milestones
ACPM	achieved critical path milestones
ACWP	actual cost of work performed
ADRLI	actual DRLIs
AEVR	actual earned rate
AHW	actual delivered hardware subsystems or units
AM	achieved milestones
AMR	actual major reviews
AQPR	actual quality point review
ASI	actual staff resource units
ASLOC	actual delivered source lines of code
ASRU	actual staff resource unit
AT	acceptance testing
ATH	actual training hours
AU	actual units
BCWP	budgeted cost of work performed
BCWP-Cum	cumulative budgeted cost of work performed
BCWS	budgeted cost of work scheduled
BPI	budget performance index
BU	budgeted units
CAM	cumulative achieved milestones
CASE	Coordination and systems engineering
CAR	cumulative actual risk (Sum of IDR_i
CAU	cumulated actual units
CBPI	cumulative budget performance index
CBU	cumulative budgeted units
CMPI	cumulative milestone performance index
COTR	contracting officer technical representative
COTS	commercial off the shelf
CPM	critical path method
CPI	cost performance index
CPPI	cumulative project performance index
CR	change request
CSI	cost schedule index
CSM	cumulative scheduled milestones
CV	cost variance
DR	discrepancy report
DRD	data requirements description
DRL	data requirements list
DRLIs	data requirements list items

DRLID	data requirements list items delivered (N_i)
DRLRA	data requirements list (DRLs) rework average
DRR	DR rate
EBT	estimated best time
EV	earned value
EVCRI	earned value rate to completion rate index
EWT	estimated worst time
GFE	government furnished equipment
HWDRR	hardware development DRs rate
HWUT	hardware units in test or operating at time of testing
IDR_i	Identified risk per reporting period
K_1	weighted relative importance of hardware
K_2	weighted relative importance of software
KSLOC	thousands of source lines of code
MAM	moved, added, and deleted milestones
MEVR	minimum earned value rate
MPI	milestone performance index
MRK	mitigated risk
MT	mode time
MVYI	milestone volatility for the next year
PAT	projected activity time
PCPI	project critical path performance index
PDRLI	planned DRLIs
PERT	program evaluation and review technique
PHW	planned hardware subsystems or units
PMR	planned major reviews
PPI	project performance index
PQPR	planned quality point review
PRUN	project unit
PSLOC	planned source lines of code
PSI	planned staff resource units
PSR	project status review
PSRU	planned staff resource unit

PTH	planned training hours
QPS	quality point scores
QT	qualification testing
RC	revealed requirements change
RID	review item disposition
RRK	remaining risk
RTP	remaining time period
RVBI	requirements volatility from baseline index
SAM	scheduled acquisition milestones
SCPM	scheduled critical path milestones
SDCP	schedule deviation on the critical path
SDDR	system detailed design review
SEO	system engineering office
SFDR	system functional design review
SM	scheduled milestones
SOP	standard operating procedure
SPI	schedule performance index
SRR	system requirements review
SSAT	subsystem acceptance test
SSCDR	subsystem critical design review
SSIT	software in KSLOC in test
SSPDR	subsystem preliminary design review
SV	schedule variance
SWDR	software DRs
SWDRR	software development DRs rate
TM	total milestones
TR	test run
TRK	total risk

D.3 Metrics Data Collection Forms

The data collection forms cited in section 8.4 on page 41 are presented here in Tables 8 through 11. They will serve the early stages of next year's RTOP task by organizing a unified approach to the collection of data in the development environment. It is expected that the manual data collection process, which these forms support, will yield to an automated electronic process as the metrics support program matures.

Table 8. RDL METRICS DATA INPUT SHEET

RDL METRICS DATA INPUT SHEETS

Project - **WBS ID -** **Report Period -**

	Milestones Month-Year	Jul 96	Aug 96	Sep 96	Oct 96	Nov 96	Dec 96	Jan 97	Feb 97	Mar 97	Apr 97	May 97	Jun 97	Jul 97	Aug 97	Sep 97	Oct 97	Nov 97	Dec 97
Software Size (KSLOC)	Baseline																		
	Planned New																		
	Planned Modified																		
Language	Reuse																		
	Deleted																		
	Unmade Baseline																		
	Total Plan Delve																		
	Comments Actual																		
	Comments Ratio																		
	Actual New																		
	Actual Modified																		
	Actual Named																		
	Actual Cum N+M																		
	Total Actual																		
Software Size (Function Points)	Baseline																		
	Planned New																		
	Planned Modified																		
	Reuse																		
Language	Deleted																		
	Unmade Baseline																		
	Total Plan Delve																		
	Comments Actual																		
	Comments Ratio																		
	Actual New																		
	Actual Modified																		
	Actual Named																		
	Actual Cum N+M																		
	Total Actual																		
Reusability	No Generic Units																		
	Instances																		
	Generic Unit																		

Table 9. RDL METRICS DATA INPUT SHEETS (Continued)

RDL METRICS DATA INPUT SHEETS (Continued)

Project - **WBS ID -** **Report Period -**

	Milestones Month-Year	Jul 96	Aug 96	Sep 96	Oct 96	Nov 96	Dec 96	Jan 97	Feb 97	Mar 97	Apr 97	May 97	Jun 97	Jul 97	Aug 97	Sep 97	Oct 97	Nov 97	Dec 97
Software Staff Hours																			
	Planned																		
	Actual																		
	OTC																		
	Planned Cum																		
	Actual Cum																		
	OTC Cum																		
Software Reqts Stability																			
	Total Reqts																		
	RC Cum																		
	DR Cum																		
Dev Progress																			
	Des - Planned																		
	Des - Actual																		
	Code - Planned																		
	Code - Actual																		
	UT - Planned																		
	UT - Actual																		
	UIT - Planned																		
	UIT - Actual																		
Computer Resource Utilization																			
	CPU - Util																		
	CPU - Cap																		
	Memory -Util																		
	Memory - Cap																		
	Disk - Util																		
	Disk - Cap																		
	I/O Ch - Util																		
	I/O Ch - Cap																		

Table 10. RDL METRICS DATA INPUT SHEETS (Continued)

RDL METRICS DATA INPUT SHEETS (Continued)

Project - **WBS ID -** **Report Period -**

Milestones Month-Year		Jul 96	Aug 96	Sep 96	Oct 96	Nov 96	Dec 96	Jan 97	Feb 97	Mar 97	Apr 97	May 97	Jun 97	Jul 97	Aug 97	Sep 97	Oct 97	Nov 97	Dec 97
DR Open Duration	Crit 0 -9 days																		
	10-29																		
	30-59																		
	60 +																		
	Maj 0 -9 days																		
	10-29																		
	30-59																		
	60 +																		
	Min 0 -9 days																		
	10-29																		
	30-59																		
	60 +																		
RC Open Duration	Crit 0 -9 days																		
	10-29																		
	30-59																		
	60 +																		
	Maj 0 -9 days																		
	10-29																		
	30-59																		
	60 +																		
	Min 0 -9 days																		
	10-29																		
	30-59																		
	60 +																		
Design Complexi-ty	Tot Subprograms																		
	No w/McCabe>10																		
	No w/McCabe>40																		
Test Case Comple-tion	SITs Planned																		
	SITs Actual																		

Table 11. RDL METRICS DATA INPUT SHEETS (Continued)

RDL METRICS DATA INPUT SHEETS (Continued)

Project - **WBS ID -** **Report Period -**

	Milestones Month-Year	Jul 96	Aug 96	Sep 96	Oct 96	Nov 96	Dec 96	Jan 97	Feb 97	Mar 97	Apr 97	May 97	Jun 97	Jul 97	Aug 97	Sep 97	Oct 97	Nov 97	Dec 97
Fault Density	Active Test Hours																		
Total DRs																			
- - - - - - - -	Opened Cum																		
• Fault	Critical																		
Density	Major																		
	Minor																		
• Test	Closed Cum																		
Focus	Critical																		
• Software	Major																		
Reliability	Minor																		
	Closed w/Sgl Fix																		
	Critical																		
	Major																		
	Minor																		
	Backlog																		
	Critical																		
	Major																		
	Minor																		
Total RCs																			
- - - - - - - -	Opened Cum																		
• Fault	Critical																		
Density	Major																		
	Minor																		
• Test	Closed Cum																		
Focus	Critical																		
• Software	Major																		
Reliability	Minor																		
	Closed w/Sgl Fix																		
	Critical																		
	Major																		
	Minor																		
	Backlog																		
	Critical																		
	Major																		
	Minor																		